A decomposition method for finding optimal container stowage plans

Roberti, R.; Pacino, D.

published in
Transportation Science
2018

DOI (link to publisher)
10.1287/trsc.2017.0795

document version
Publisher's PDF, also known as Version of record

document license
Article 25fa Dutch Copyright Act

Transportation Science

Publication details, including instructions for authors and subscription information:
http://pubsonline.informs.org

A Decomposition Method for Finding Optimal Container Stowage Plans

R. Roberti, D. Pacino

To cite this article:
R. Roberti, D. Pacino (2018) A Decomposition Method for Finding Optimal Container Stowage Plans. Transportation Science 52(6):1444-1462. https://doi.org/10.1287/trsc.2017.0795

http://pubsonline.informs.org/journal/trsc/

TRANSPORTATION SCIENCE
Vol. 52, No. 6, November–December 2018, pp. 1444–1462
ISSN 0041-1655 (print), ISSN 1526-5447 (online)

A Decomposition Method for Finding Optimal Container Stowage Plans

R. Roberti,[a] D. Pacino[b]

[a] Department of Information, Logistics, and Innovation, Vrije Universiteit Amsterdam, 1081 HV Amsterdam, Netherlands;
[b] Department of Management Engineering, Technical University of Denmark, 2800 Kongens Lyngby, Denmark

Contact: r.roberti@vu.nl, http://orcid.org/0000-0002-2987-1593 (RR); darpa@dtu.dk (DP)

Received: January 23, 2017
Revised: May 26, 2017
Accepted: July 16, 2017
Published Online in Articles in Advance: November 1, 2018

https://doi.org/10.1287/trsc.2017.0795

Abstract. In transportation of goods in large container ships, shipping industries need to minimize the time spent at ports to load/unload containers. An optimal stowage of containers on board minimizes unnecessary unloading/reloading movements, while satisfying many operational constraints. We address the basic *container stowage planning problem* (CSPP). Different heuristics and formulations have been proposed for the CSPP, but finding an optimal stowage plan remains an open problem even for small-sized instances. We introduce a novel formulation that decomposes CSPPs into two sets of decision variables: the first defining how single container stacks evolve over time and the second modeling port-dependent constraints. Its linear relaxation is solved through stabilized column generation and with different heuristic and exact pricing algorithms. The lower bound achieved is then used to find an optimal stowage plan by solving a mixed-integer programming model. The proposed solution method outperforms the methods from the literature and can solve to optimality instances with up to 10 ports and 5,000 containers in a few minutes of computing time.

Funding: This work is part of the DynaStow (Dynamic Programming-based Stowage Planning) project funded by the Den Danske Maritime Fond. This support is gratefully acknowledged.
Supplemental Material: The online appendix is available at https://doi.org/10.1287/trsc.2017.0795.

Keywords: column generation • container stowage • dynamic programming • exact methods • maritime logistics • mixed-integer programming

1. Introduction

The use of containers for the transportation of consumer goods has been drastically growing since its introduction in 1956 (Levinson 2010). Liner shipping is the transportation of goods in large container ships that sail the world on fixed routes and schedules. Prior to the financial crisis in 2009, the booming demand for containerized transport drove shipping lines to deploy larger and larger ships that can now carry over 18,000 twenty-foot equivalent units (TEU). In recent years of economic recovery, the shipping industry is facing a decrease in demand and an overcapacitated network, which is now driving freight and charter rates down (UNCTAD 2015). A reduction in time spent at ports for loading/unloading is extremely important when making a profit on marginal costs. With a cargo flow of approximately 15.4 million TEU just on the Asia–Europe trade route in 2014, the industry has recognized the need to apply scientific optimization methods. This is evident from the growing literature in liner shipping network optimization, speed optimization, stowage planning, etc.

Stowage planning, which is the focus of this paper, is the process of assigning containers to positions on the ship (*stowage*). The order in which containers are stowed and their distribution along the ship define the cost of the plan. The arrangement of the containers can reduce the time at port by minimizing the number of unnecessary container moves and by optimizing the schedule of the loading cranes (*quay cranes*). Moreover, it contributes to the total bunker consumption since it can impact the hydrodynamics of the ship (e.g., *trim optimization*).

Finding high-quality stowage plans is hard not only because of the complexity of the objective function but also because of the complex set of feasibility requirements to satisfy. Until now, the academic focus has been on developing heuristic methods that have gradually moved from simple capacity allocation problems (Avriel and Penn 1993; Dubrovsky, Levitin, and Penn 2002; Ambrosino, Sciomachen, and Tanfani 2004) to handling the complex stability requirements of a ship (Pacino et al. 2011, 2012). However, the quality of the implemented solutions is hard to scientifically evaluate because of the lack of studies on optimal methods and the unwillingness of the industry to share data.

In this paper, we aim to (partially) fill this gap by presenting insights and solution methods to solve to optimality the basic *container stowage planning problem* (CSPP) originally defined by Avriel and Penn (1993), which is one of the first stowage planning problems that appeared in the literature and can be described as

follows. Containers are stowed on a ship divided into bays. Each bay consists of multiple stacks, and each stack is divided into tiers. Each tier can contain a single container. Moreover, all containers are of the same size, all stacks have the same number of tiers, and all bays have the same number of stacks. Therefore, the ship can be seen as a grid (i.e., a single rectangular bay), where each column represents a stack and each row represents a tier.

The ship travels a predefined route of ports, where containers are loaded and unloaded. Containers can be accessed only from the top of each stack, and each container has an origin port and a destination port. The number of containers to move from each port to any other port is known in advance; a pair of origin/destination ports is commonly called a *transport*. The ship is empty before loading operations take place at the first port and after it arrives at the last port. At each port, containers destined to that port are unloaded (such unloading operations are called *discharges*), and containers destined to the following ports are loaded. Without loss of generality, it can be assumed that all unloading operations take place before any loading operation.

Whenever a container has to be discharged, another container destined to one of the next ports may be stowed on top of it: this situation is called *overstowage* and requires a *shift* (i.e., the unloading and reloading of the container on top). Two types of shifts may occur: *mandatory* and *nonmandatory*. Mandatory shifts take place whenever there is overstowage. Nonmandatory shifts take place if a container is unloaded and reloaded at a port to avoid overstowage.

An example of mandatory and nonmandatory shifts is provided in Figure 1. The left panel shows two stacks with five tiers each and the containers stowed upon arriving in Port 4 (each container is identified by its destination port). Containers destined to Port 4 must be discharged, and four containers destined to Port 6 must be loaded. The layout of the containers in the two stacks is shown in the right panel; six shifts have taken place for the containers destined to Port 5: four mandatory shifts in the stack on the left and two nonmandatory shifts in the stack on the right.

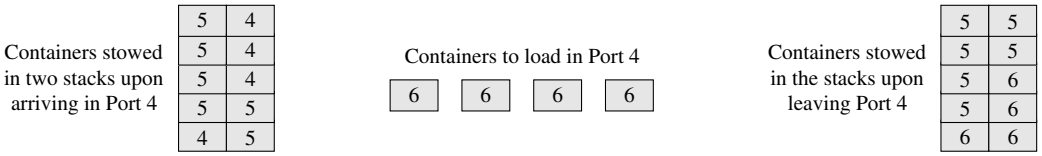
In the CSPP, both mandatory and nonmandatory shifts are allowed. Note that nonmandatory shifts are sometimes called *voluntary* in the literature; nonetheless, in real-life applications, voluntary shifts refer to shifts performed by terminal operators that were not planned in advance, so we prefer to use the term nonmandatory.

The goal of the CSPP is to find a stowage plan where each container is transported from its origin port to its destination port and the number of shifts is minimized.

The CSPP is a simplified version of stowage planning problems encountered in practice, where, forexample, containers may have different sizes and weights, stacks may have a different number of tiers, and stability constraints on the distribution of weight along the ship must be taken into account. In spite of these simplifications, the CSPP has features that are common to all real-life stowage planning problems, so effective solution methods for the CSPP can be the basis of new approaches for solving more complex stowage planning problems. Although several exact and heuristic methods for the CSPP can be found in the literature, solving to optimality large-scale CSPP instances using these methods is still out of reach even with today's technology. Therefore, this paper provides more insight on the CSPP and on solving it to optimality. In particular, the main contributions of this paper are the following:

- By extending previous results from the literature, we derive some properties on the transport matrix guaranteeing that a strictly positive number of shifts must take place and propose a combinatorial lower bound on the minimum number of shifts.
- We introduce a novel *mixed-integer programming* (MIP) formulation for the CSPP, based on two sets of exponentially many variables, whose linear relaxation provides strong lower bounds.
- We illustrate a lower bounding procedure based on stabilized *column generation* (CG) and multiple heuristic and exact pricing algorithms to solve the linear relaxation of the new formulation.
- We describe a solution method combining the lower bound from the bounding procedure with a compact MIP model to determine an optimal stowage plan.
- We computationally prove that the proposed solution method can provide an optimal stowage plan with no shifts on all test instances with up to 10 ports and 5,000 TEU from the literature in a few minutes of computing time.
- We generate a new set of test instances, where shifts must take place, and show that they are computationally harder, but the proposed solution method can

Figure 1. An Example of Mandatory and Nonmandatory Shifts



still solve instances with up to 10 ports and 5,000 TEU, performing significantly better than the MIP models from the literature.

The paper is organized as follows. Section 2 reviews the literature about stowage planning. The CSPP is formally introduced in Section 3. A novel mathematical model is illustrated in Section 4. Section 5 provides two MIP models to formulate the pricing problems of the variables of the new formulation. Section 6 describes the lower bounding procedure, and Section 7 illustrates the procedure for finding an optimal stowage plan. Computational results are reported in Section 8. Finally, some conclusions and future research directions are outlined in Section 9.

2. Literature Review

Literature on stowage planning is very diverse, in terms of both problem definitions and solution approaches. The lack of a common benchmark and the unwillingness from the industry to share data could be cited as probable causes. The literature can be roughly divided into two major groups: those works addressing the CSPP studied in this paper and those that include a rich set of characteristics.

2.1. Literature on the CSPP

Solution methods for the CSPP have been presented by Avriel and Penn (1993), Avriel et al. (1998), Dubrovsky, Levitin, and Penn (2002), and Ding and Chou (2015). Avriel and Penn (1993) were the first to propose a binary linear formulation for the CSPP; they used two sets of variables, the first one with five indexes (three for the ports, one for the tiers, and one for the stacks) and the second with three indexes (for each port, each tier, and each stack). As the number of variables easily grows with the size of the instance, they also proposed a heuristic method based on applying the proposed binary linear model on a reduced transportation matrix. Avriel et al. (1998) presented the *suspensory heuristic* (a dynamic slot-assignment scheme) and Dubrovsky, Levitin, and Penn (2002) a genetic algorithm with a compact encoding for the same problem. A different binary formulation has recently been proposed by Ding and Chou (2015), which is reported to solve in 10s of minutes small-size problems with seven ports, six tiers, and 150 stacks; however, the core of the contribution is a novel placement heuristic that generally performs better than the suspensory heuristic.

The complexity of the CSPP was addressed by Aslidis (1990), Avriel, Penn, and Shpirer (2000), and Tierney, Pacino, and Jensen (2014). Aslidis (1990) presented a polynomial-time algorithm for solving the CSPP with a *single* column. Avriel, Penn, and Shpirer (2000) studied its connection with problems of coloring a circle graph and proved that the problem is NP-complete. Tierney, Pacino, and Jensen (2014) presented a polynomial-time algorithm for the CSPP with fixed stacks and tiers, demonstrating that the combinatorial complexity is not rooted in the number of transported containers; moreover, they showed that a version of the problem where tiers are abstracted away continues to be NP-hard.

2.2. Literature on Rich Stowage Planning Problems

Rich problem definitions tend to include various combinations of stowage requirements. Focusing on a single port problem with heuristic stability rules and a mixture of container types, Ambrosino, Sciomachen, and Tanfani (2004) proposed a binary formulation for the *master bay planning problem* (MBPP). The problem is solved using a heuristic preprocessing procedure, which is then extended to a three-phase heuristic by Ambrosino, Sciomachen, and Tanfani (2006) and then further combined with a tabu search by Ambrosino et al. (2009). The relation between the MBPP and the 3-*Dimensional Bin Packing Problem* is studied by Sciomachen and Tanfani (2003, 2007), where the ship is seen as the only available bin. A constraint programming model and a constraint-based local search procedure for a single bay stowage can also be found in Delgado et al. (2012) and Pacino et al. (2012), respectively.

The first rich multiport stowage planning model was presented by Botter and Brinati (1992); the developed binary formulation was too complex, so a tree search procedure that could solve instances with four ports and 40 TEU was implemented. The most successful heuristics are based on a hierarchical decomposition of the problem where a *master planning* phase distributes containers to sections of the vessel and successively a *slot planning* phase assigns containers to those sections. Examples of such decompositions can be found in Wilson and Roach (1999), Kang and Kim (2002), Pacino et al. (2011), Ambrosino, Paolucci, and Sciomachen (2017), and Ambrosino, Paolucci, and Sciomachen (2015). All of them proposed a mathematical formulation for the first phase and a heuristic for the second phase. A study on the linearization of complex vessel stability calculations is also presented in Pacino and Jensen (2013).

3. Problem Definition

In this section, we formally introduce the CSPP along with the notation used throughout the paper and report some interesting properties that can be found in the literature.

As previously noted, the ship can be seen as a grid. We denote with C the set of stacks ($|C| = \bar{c}$) and with R the set of tiers ($|R| = \bar{r}$), which correspond to the columns and the rows of the grid, respectively. Tiers are numbered from 1 to $\bar{r}$ from bottom to top. Let R^- be the set of all tiers excluding the top one (i.e., $R^- = \{1, 2,$

$\ldots, \bar{r}-1\}$). The ship sequentially visits the ordered set of ports $P = \{1, 2, \ldots, n\}$ from 1 to n.

Let t_{ij} be the number of containers originating from port i and destined to port j, with $1 \le i < j \le n$ ($t_{ij} = 0$ if $i \ge j$), and let $\mathbf{T} = [t_{ij}] \in \mathbb{Z}_+^{n \times n}$ be the transportation matrix. Moreover, let $\mathcal{T}$ be the set of all transports (i.e., $\mathcal{T} = \{(i,j)\colon i, j \in P, t_{ij} > 0\}$. A container destined to Port j, no matter its origin port, is called a *j-container*. For the sake of brevity, the subset of ports $\{i, i+1, \ldots, j\}$, with $1 \le i < j \le n$, is indicated with P_i^j. Moreover, let $\hat{\mathcal{T}} = \{(i,j)\colon i \in P_1^{n-1}, j \in P_{i+1}^n\}$ be the set of all pairs of ports (i,j) with $i < j$, let $n\,\text{Cont} = \sum_{(i,j)\in\mathcal{T}} t_{ij}$ be the number of containers transported (also equal to the total number of discharges), and let $\hat{t}_{ij}$ be the number of j-containers to stow on board upon leaving Port i (i.e., $\hat{t}_{ij} = \sum_{k \in P_1^i} t_{kj}$), for each $(i,j) \in \hat{\mathcal{T}}$.

The transportation matrix $\mathbf{T}$ is *feasible* if all containers can be stowed on board upon leaving each port $i \in P_1^{n-1}$, that is, if $\sum_{j=i+1}^n \hat{t}_{ij} \le \bar{r} \cdot \bar{c}$.

Ding and Chou (2015) reported and proved the following properties of the CSPP.

Property 1. *If $n \le 3$, for any given $\bar{r}$, $\bar{c}$, and $\mathbf{T}$, there always exists a stowage plan with no shifts.*

Property 2. *If $\bar{r} = 1$, for any given n, $\bar{c}$, and $\mathbf{T}$, all stowage plans lead to no shifts.*

Property 3. *If $n \ge 4$ and $\bar{r} \ge 2$, for any given $\bar{c}$, there always exists a transportation matrix $\mathbf{T}$ such that shifts are inevitable.*

In particular, Property 3 was proven on the following transportation matrix:

$$\mathbf{T} = \begin{bmatrix} 0 & 1 & \bar{c}\cdot\bar{r}-1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix},$$

where $n = 4$, $t_{12} = 1$, $t_{13} = \bar{c}\cdot\bar{r} - 1$, and $t_{24} = 1$. A transportation matrix $\mathbf{T}$ where shifts are inevitable if $n > 4$ can easily be derived by extending the previous one. In Section 6.2, we build on Property 3 to compute a combinatorial lower bound on the optimal number of shifts.

A transportation matrix $\mathbf{T}$ is defined as *full-loading* if the ship is always full upon leaving each port (i.e., $\sum_{j=i+1}^n \hat{t}_{ij} = \bar{r}\cdot\bar{c}$, for each $i \in P_1^{n-1}$); otherwise, $\mathbf{T}$ is said to be *nonfull loading*. Clearly, a full-loading transportation matrix is also feasible. Ding and Chou (2015) showed that a nonfull-loading transportation matrix $\mathbf{T}'$ can be transformed into a full-loading transportation matrix $\mathbf{T}$ without affecting the number of shifts in the optimal solution as follows: $t_{ij} = t'_{ij}$ if $j \ne i+1$, and $t_{ij} = t'_{ij} + \bar{r}\cdot\bar{c} - \sum_{k=1}^i \sum_{j=i+1}^n t'_{kj}$ if $j = i+1$. Therefore, without loss of generality, we consider full-loading matrices only.

In the remainder of the paper, the symbols "$\wedge$," "$\vee$," and "mod" indicate the logical-and, logical-or, and modulo operation, respectively, and bold symbols represent vectors or matrices. Moreover, the notation introduced thus far is summarized in Table 1.

Table 1. Notation Used Throughout the Paper

Symbol	Meaning
n	Number of ports to visit
P	Set of ports $P = \{1, \ldots, n\}$
P_i^j	Subset of ports from i to j with $i < j$, i.e., $P_i^j = \{i, i+1, \ldots, j\}$
C	Set of stacks
$\bar{c}$	Number of stacks $\bar{c} = \lvert C\rvert$
R	Set of tiers
$\bar{r}$	Number of tiers per stack $\bar{r} = \lvert R\rvert$
R^-	Set of tiers excluding the top one, i.e., $R^- = \{1, 2, \ldots, \bar{r}-1\}$
$\mathbf{T}$	Transportation matrix
t_{ij}	Transport from Port i to Port j with $1 \le i < j \le n$
$\mathcal{T}$	Set of transports, i.e., $\mathcal{T} = \{(i,j)\colon i, j \in P, t_{ij} > 0\}$
$\hat{\mathcal{T}}$	Set of pairs of ports (i,j) with $1 \le i < j \le n$
$\hat{t}_{ij}$	Number of j-containers to stow on board upon leaving Port i with $(i,j) \in \hat{\mathcal{T}}$
$n\,\text{Cont}$	Total number of containers to transport along the route

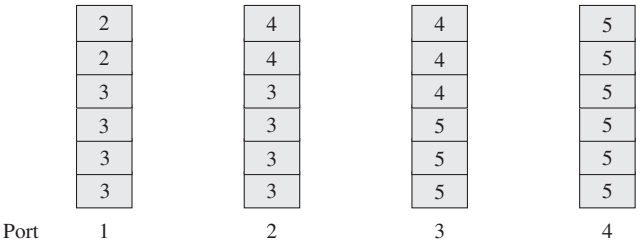
4. A Novel Mathematical Formulation

This section introduces a novel mathematical formulation of the CSPP. It is based on two sets of exponentially many variables. The first set corresponds to *stack plans*, which define how containers stowed in a given stack are moved while visiting the different ports. The second set corresponds to *port layouts*, which indicate how containers are stowed on the ship upon leaving a given port. The linear relaxation of the formulation provides strong lower bounds (as will be shown in Section 8) but needs CG to be solved. An effective lower bounding procedure will be presented in Section 6.

Let $\mathcal{S}$ be the index set of all feasible stack plans for any stack $c \in C$, where each stack plan s is represented by an $\bar{r}\lvert\hat{\mathcal{T}}\rvert$-dimensional vector $\mathbf{e}^s \in \{0,1\}^{\bar{r}\lvert\hat{\mathcal{T}}\rvert}$, where each element $e^s_{ijr} \in \{0,1\}$ indicates if, upon leaving Port i, exactly r j-containers are stowed in the stack (if so, $e^s_{ijr} = 1$). The cost, $\tilde{c}_s$, of stack plan $s \in \mathcal{S}$ is the minimum number of unloading operations (i.e., discharges plus mandatory and nonmandatory shifts) required to stow re^s_{ijr} j-containers in the stack upon leaving port i (for each $(i,j) \in \hat{\mathcal{T}}$ and each $r = 1, \ldots, \bar{r}$). A stack plan s is feasible if, upon leaving each port $i \in P_1^{n-1}$, exactly $\bar{r}$ containers are stowed into it (i.e., $\sum_{j=i+1}^n \sum_{r\in R} re^s_{ijr} = \bar{r}$ for each $i \in P_1^{n-1}$).

An example of a stack plan is provided in Figure 2. A stack plan shows which containers are stowed in a given stack upon leaving each port from Port 1 to Port $n-1$. The figure shows an example of a feasible stack plan s for an instance with five ports and six tiers per stack. The cost $\tilde{c}_s$ of this stack plan is 17 because of two discharges in Port 2, four discharges and two mandatory shifts in Port 3, three discharges in Port 4, and six

Figure 2. An Example of a Stack Plan



discharges in Port 5; the stack plan is represented as $e^s_{122} = e^s_{134} = e^s_{234} = e^s_{242} = e^s_{343} = e^s_{353} = e^s_{456} = 1$ and all other elements e^s_{ijr} equal to 0.

Let $\mathscr{L}_i$ be the index set of all feasible port layouts for port $i \in P_1^{n-1}$, where each port layout l is represented by an $\bar{r}|P^n_{i+1}|$-dimensional vector $\mathbf{a}^{il} \in \mathbb{Z}_+^{\bar{r}|P^n_{i+1}|}$, where each element $a^{il}_{jr} \in \mathbb{Z}_+$ indicates the number of stacks that, upon leaving port i, contain r $(1 \le r \le \bar{r})$ j-containers. A port layout $l \in \mathscr{L}_i$ is feasible if all j-containers are stowed on board (i.e., $\sum_{r \in R} r a^{il}_{jr} = \hat{t}_{ij}$ for each port $j \in P^n_{i+1}$).

An example of a port layout is provided in Figure 3. A port layout shows how containers are stowed in the different stacks upon leaving a given port. The figure illustrates an example of a feasible port layout l for Port 1 for an instance with five ports, five stacks, six tiers per stack, and transports $t_{12} = 5$, $t_{13} = 6$, $t_{14} = 7$, $t_{15} = 12$; the port layout is represented as $a^{1l}_{22} = 1$, $a^{1l}_{23} = 1$, $a^{1l}_{33} = 2$, $a^{1l}_{41} = 1$, $a^{1l}_{46} = 1$, $a^{1l}_{53} = 2$, $a^{1l}_{56} = 1$ and all other elements a^{1l}_{jr} equal to 0.

Let $x_s \in \mathbb{Z}_+$ be a nonnegative integer variable representing the number of stacks implementing stack plan $s \in \mathscr{S}$ over the route, and let $y_{il} \in \{0,1\}$ be a binary variable equal to 1 if port layout $l \in \mathscr{L}_i$ of port $i \in P_1^{n-1}$ is selected (0 otherwise). The CSPP can be formulated as

$$z_{\text{RP}} = \min \sum_{s \in \mathscr{S}} \tilde{c}_s x_s \tag{1}$$

$$\text{s.t.} \quad \sum_{l \in \mathscr{L}_i} y_{il} \ge 1, \quad i \in P_1^{n-1}, \tag{2}$$

$$\sum_{s \in \mathscr{S}} e^s_{ijr} x_s \ge \sum_{l \in \mathscr{L}_i} a^{il}_{jr} y_{il}, \quad (i,j) \in \hat{\mathscr{T}}, r \in R, \tag{3}$$

$$x_s \in \mathbb{Z}_+, \quad s \in \mathscr{S}, \tag{4}$$

$$y_{il} \in \{0,1\}, \quad i \in P_1^{n-1}, l \in \mathscr{L}_i. \tag{5}$$

Figure 3. An Example of a Port Layout

2	3	2	4	5
2	3	2	4	5
4	3	2	4	5
5	5	3	4	5
5	5	3	4	5
5	5	3	4	5

The objective function (1) aims to minimize the total number of unloading operations in the selected stack plans. Constraints (2) require selecting at least one port layout for each port $i \in P_1^{n-1}$. Constraints (3) link variables $\mathbf{x}$ and $\mathbf{y}$ by guaranteeing that, if in the selected port layout for port i there are a^{il}_{jr} columns containing r j-containers, then at least a^{il}_{jr} stack plans containing r j-containers are also selected. Constraints (4)–(5) ensure integrality of the decision variables.

Note that constraints (2)–(3) will be tight at any optimal solution of (1)–(5). Because of the exponential size of both sets of variables, CG is needed to solve the linear relaxation of formulation (1)–(5), and having (2)–(3) in the form of inequalities halves the feasible space of dual variables, thus generally improving the convergence of solution methods based on CG.

Moreover, the formulation could arguably be extended to rich variants of the CSPP by adding some stack-related and port-related constraints to the definition of feasible stack plans and port layouts, respectively.

In Sections 5–7, we describe the different ingredients that allow to find an optimal CSPP solution in two subsequent steps. In the first step, a lower bound LB on the minimum number of unloading operations, corresponding to the optimal solution cost of the linear relaxation of formulation (1)–(5), is computed by CG. In the second step, an optimal CSPP is found by solving one of the two MIP models with a general-purpose MIP solver. The first MIP model (described in Section 7.1) is used to find CSPP solutions with no shifts. The second MIP model (described in Section 7.2) allows shifts in a small subset of stacks. The choice between which of the two MIP models to use is based on the lower bound LB computed in the first step: if LB = n Cont, then a stowage plan with no shifts is likely to exist, so the first MIP model is selected; otherwise (i.e., if LB > n Cont), at least LB – n Cont shifts are necessary, so the second MIP model is selected and shifts are allowed in LB – n Cont stacks only.

In the first step, to compute lower bound LB by CG, two pricing problems (for the stack plans and for the port layouts) have to be solved at each iteration. We introduce an MIP model to solve the pricing problem for stack plans in Section 5.1, and an MIP model to solve the pricing problem for port layouts in Section 5.2. Because it would be time-consuming to solve both pricing problems via these MIP models at each iteration, we also introduce two heuristic procedures based on dynamic programming to generate stack plans (see Sections 6.4.1–6.4.2) and a heuristic procedure to generate port layouts (see Section 6.4.3). We also describe a combinatorial lower bound (see Section 6.2) that can be used to limit the number of iterations of CG, and a stabilization technique (see Section 6.3) to limit degeneracy of the dual variables.

5. Solving the Linear Relaxation of the New Formulation

We solve the linear relaxation of the formulation (1)–(5) via CG. There are two pricing problems: one for stack plans and another for port layouts. In this section, we formulate these two pricing problems as MIP problems.

The goal of CG is to solve the linear relaxation of (1)–(5) obtained by replacing constraints (4) with $x_s \geq 0$ ($s \in \mathcal{S}$) and constraints (5) with $y_{il} \geq 0$ ($l \in \mathcal{L}_i$, $i \in P_1^{n-1}$)—note that constraints $y_{il} \leq 1$ are redundant and can be omitted. Hereafter, we call this problem the *master problem* (MP), and we denote its optimal value by z_{MP}. Because of the size of the sets $\mathcal{S}$ and $\mathcal{L}_i$, CG works on a *restricted master problem* (RMP) containing small subsets $\hat{\mathcal{S}} \subseteq \mathcal{S}$ and $\hat{\mathcal{L}}_i \subseteq \mathcal{L}_i$ of variables of the MP. The optimal value of the RMP is denoted by z_{RMP}.

Let $u_i \in \mathbb{R}_+$ be the dual variable associated with constraint (2) for port $i \in P_1^{n-1}$, and $v_{ijr} \in \mathbb{R}_+$ be the dual variable of constraint (3) for the pair of ports $(i,j) \in \hat{\mathcal{T}}$ and number of tiers $r \in R$. The dual (hereafter called MD) of the RMP reads as follows:

$$\max \sum_{i=1}^{n-1} u_i$$

$$\text{s.t.} \sum_{(i,j)\in\hat{\mathcal{T}}} \sum_{r\in R} e^s_{ijr} v_{ijr} \leq \tilde{c}_s \quad s \in \hat{\mathcal{S}},$$

$$u_i - \sum_{j=i+1}^{n} \sum_{r\in R} a^{il}_{jr} v_{ijr} \leq 0, \quad i \in P_1^{n-1},\ l \in \hat{\mathcal{L}}_i,$$

$$u_i \in \mathbb{R}_+, \quad i \in P_1^{n-1},$$

$$v_{ijr} \in \mathbb{R}_+, \quad (i,j) \in \hat{\mathcal{T}},\ r \in R.$$

At each iteration of a basic CG procedure, the RMP is solved, the optimal dual variables $(\mathbf{u}, \mathbf{v})$ are retrieved, and the pricing problems for stack plans and for port layouts are solved to find the least-cost variables of the MP. If such columns have a negative reduced cost, they are added to the RMP, and the process is repeated. The procedure stops as soon as both pricing problems do not return any negative reduced cost columns. The two pricing problems can be formulated and solved as MIP problems as illustrated next.

5.1. An MIP Model to Price Out Stack Plans

The pricing problem for stack plans can be formulated as an MIP with the following three sets of decision variables: $\alpha_{ijr} \in \{0,1\}$, which equals 1 if, upon leaving port $i \in P_1^{n-1}$, a j-container ($j \in P_{i+1}^n$) is stowed in tier $r \in R$; $\beta_{ijr} \in \{0,1\}$, which equals 1 if, upon leaving port $i \in P_1^{n-1}$, r j-containers ($1 \leq r \leq \bar{r}$, $j \in P_{i+1}^n$) are stowed in the stack; and $\xi_{ir} \in \{0,1\}$, which equals 1 if the container stowed in tier $r \in R$ upon leaving port $i \in P_1^{n-1}$ is unloaded in port $i+1$. Then, the pricing problem for stack plans can be formulated as follows:

$$\min \left\{ \sum_{i=1}^{n-1} \sum_{r\in R} \xi_{ir} - \sum_{(i,j)\in\hat{\mathcal{T}}} \sum_{r\in R} v_{ijr} \beta_{ijr} \right\} \tag{6}$$

$$\text{s.t.} \sum_{(i,j)\in\hat{\mathcal{T}}} \alpha_{ijr} = 1, \quad i \in P_1^{n-1},\ r \in R, \tag{7}$$

$$\sum_{r\in R} \alpha_{ijr} = \sum_{r\in R} r\beta_{ijr}, \quad (i,j) \in \hat{\mathcal{T}}, \tag{8}$$

$$\sum_{(i,k)\in\hat{\mathcal{T}}: k\neq j} \beta_{ik\bar{r}} + \sum_{r\in R} \beta_{ijr} \leq 1, \quad (i,j) \in \hat{\mathcal{T}}, \tag{9}$$

$$\xi_{ir} \geq \alpha_{ijr} - \alpha_{i+1,jr}, \quad (i,j) \in \hat{\mathcal{T}},\ r \in R, \tag{10}$$

$$\xi_{ir} \leq \xi_{i,r+1}, \quad i \in P_1^{n-1},\ r \in R^-, \tag{11}$$

$$\beta_{ijr} = 0, \quad (i,j) \in \hat{\mathcal{T}},\ R \ni r > \hat{t}_{ij}, \tag{12}$$

$$\alpha_{ijr}, \beta_{ijr} \in \{0,1\}, \quad (i,j) \in \hat{\mathcal{T}},\ r \in R, \tag{13}$$

$$\xi_{ir} \in \{0,1\}, \quad i \in P_1^{n-1},\ r \in R. \tag{14}$$

The objective function (6) aims to minimize the reduced cost of the generated stack plan, given by the number of unloading operations minus the associated dual variables $\mathbf{v}$. Constraints (7) stipulate that exactly one container must be stowed in tier $r \in R$ upon leaving port $i \in P_1^{n-1}$.

Constraints (8) establish the link between $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ variables: β_{ijr} is equal to 1 if r j-containers are stowed in the stack upon leaving port i. Constraints (9) ensure that at most one of the β_{ijr} variables associated to the pair of ports $(i,j) \in \hat{\mathcal{T}}$ is equal to 1. For example, let us assume that, upon leaving Port 1, three 2-containers are stowed in the stack, thus meaning that $\sum_{r\in R} \alpha_{12r} = 3$. Variable $\beta_{123} = 1$ must be set equal to 1; this is modeled with constraint (8) (i.e., $\sum_{r\in R} \alpha_{12r} = \sum_{r\in R} r\beta_{12r}$) and with constraint $\sum_{r\in R} \beta_{12r} \leq 1$. Nevertheless, $\sum_{r\in R} \beta_{12r} \leq 1$ can be lifted by adding the first summation of constraint (9) (i.e., $\sum_{(i,k)\in\hat{\mathcal{T}}: k\neq j} \beta_{ik\bar{r}}$) because, as soon as the stack is filled up with $\bar{r}$ containers destined to one of the ports $k \in P_{i+1}^n$, $k \neq j$, then no j-containers can be stowed in the stack (i.e., $\sum_{(i,k)\in\hat{\mathcal{T}}: k\neq j} \beta_{ik\bar{r}} = 1$ and $\sum_{r\in R} \beta_{ijr} = 0$).

Constraints (10) force variable ξ_{ir} to be equal to 1 whenever the container stowed in tier r upon leaving port $i \in P_1^{n-1}$ is unloaded in port $i+1$, notwithstanding its destination port. For example, consider variable ξ_{21}, which is equal to 1 if the container stowed in Port 2 in tier 1 is unloaded in Port 3. Because of constraints (7), one of the variables α_{2j1}, with $j \in P_3^n$, must be equal to 1. Let us assume that $\alpha_{241} = 1$, meaning that a 4-container is stowed in tier 1 upon leaving Port 2. If this 4-container is unloaded in Port 3, then $\alpha_{341} = 0$, thus setting ξ_{21} equal to 1 ($\xi_{21} \geq \alpha_{241} - \alpha_{341} = 1 - 0$). Otherwise, if the 4-container is not unloaded in Port 3, then $\alpha_{341} = 1$, and $\xi_{21} = 0$ ($\xi_{21} \geq \alpha_{241} - \alpha_{341} = 1 - 1$).

Constraints (11) guarantee that, if the container stowed in tier $r \in R$ upon leaving Port $i \in P_1^{n-1}$ is unloaded in port $i+1$, then all containers stowed on top of it are unloaded as well. Constraints (12) set variables β_{ijr} equal to 0 if the total number of j-containers originated from Port 1 to i (i.e., $\hat{t}_{ij}$), with $(i,j) \in \hat{\mathcal{T}}$, is less than $r \in R$. Integrality on the decision variables is imposed with constraints (13)–(14).

The minimum reduced cost stack plan $s \in \mathcal{S}$ is defined as $e^s_{ijr} = \beta^*_{ijr}$ for each $(i,j) \in \hat{\mathcal{T}}$ and each $r = 1,\dots,\bar{r}$, where $\boldsymbol{\beta}^*$ is the optimal solution of (6)–(14).

5.2. An MIP Model to Price Out Port Layouts

Let $\mathcal{H}_i$ be the index set of all feasible *stack patterns* for port $i \in P^{n-1}_1$, where each stack pattern $h \in \mathcal{H}_i$ is a subset of $\bar{r}$ containers destined to ports $i+1,\dots,n$ that can be stowed, with no overstowage, in a single stack. Each stack pattern $h \in \mathcal{H}_i$ is represented by an $(n-i)$-dimensional vector $\mathbf{w}^h \in \mathbb{Z}^{n-1}_+$ such that $\sum^n_{j=i+1} w^h_j = \bar{r}$ and $w^h_j \le \hat{t}_{ij}$ for each $j \in P^n_{i+1}$, where w^h_j is the number of j-containers stowed in the stack pattern.

Let $\chi_{ih} \in \mathbb{Z}_+$ be a nonnegative integer variable representing the number of stacks implementing stack pattern $h \in \mathcal{H}_i$ in port layout of Port $i \in P^{n-1}_1$, and let d_{ih} be the reduced cost of stack pattern $h \in \mathcal{H}_i$ computed as $d_{ih} = \sum_{j \in P^n_{i+1}: w^h_j > 0} v_{ij} w^h_j$. Then, the pricing problem for port layouts for a given port $i \in P^{n-1}_1$ can be formulated as follows:

$$\min \left\{ \sum_{h \in \mathcal{H}_i} d_{ih} \chi_{ih} - u_i \right\} \tag{15}$$

$$\text{s.t.} \sum_{h \in \mathcal{H}_i} w^h_j \chi_{ih} = \hat{t}_{ij}, \quad j \in P^n_{i+1}, \tag{16}$$

$$\sum_{h \in \mathcal{H}_i} \chi_{ih} = \bar{c}, \tag{17}$$

$$\chi_{ih} \in \mathbb{Z}_+, \quad h \in \mathcal{H}_i. \tag{18}$$

The objective function (15) minimizes the reduced cost of the optimal port layout. Constraints (16) stipulate that all transports from Port i have to be satisfied. Constraint (17) guarantees that $\bar{c}$ stack patterns are selected. Integrality on the decision variables is imposed with constraints (18).

The minimum reduced cost port layout l is defined as $a^{il}_{jr} = \sum_{h \in \mathcal{H}_i: w^h_j = r} \chi^*_{ih}$ for each $j \in P^n_{i+1}$ and $r = 1,\dots,\bar{r}$, where $\boldsymbol{\chi}^*$ is the optimal solution of (15)–(18).

Note that the formulation (15)–(18) has an exponential number of variables because the number of stack patterns of the sets $\mathcal{H}_i$ increases exponentially with the number of ports. Therefore, to use formulation (15)–(18) to solve the pricing problem for port layouts, a column generation strategy needs to be applied unless it is possible to enumerate all stack patterns (i.e., all of the variables $\boldsymbol{\chi}$) a priori. The computational experiments reported in Section 8 involve instances with up to 10 ports and up to 10 tiers per stack (i.e., $\bar{r} \le 10$), so it is possible to completely enumerate the sets $\mathcal{H}_i$ a priori. Thus, to solve the pricing problem for port layouts, we simply solve problem (15)–(18) by using a general-purpose MIP solver after numerating all of the stack patterns and the corresponding variables $\boldsymbol{\chi}$.

6. A Lower Bounding Procedure Based on the New Formulation

To efficiently solve the MP, two main tools can be used: heuristic algorithms to price out columns, and stabilization techniques to stabilize the dual variables and limit the number of iterations of the CG procedure. In this section, we outline the bounding procedure we propose to solve the MP and its main ingredients. The reader interested in a detailed description of the procedure is referred to the online appendix, where all details and a pseudo-code are provided.

The CG bounding procedure solves the RMP by starting with a small subset of variables $\mathbf{x}$ and $\mathbf{y}$ corresponding to a CSPP solution obtained via a greedy algorithm (see Section 6.1). At each iteration, stack plans of negative reduced cost are generated via two heuristic algorithms (see Sections 6.4.1 and 6.4.2), and the most negative reduced cost stack plan found by each of the two procedures is added to the RMP; as soon as none of the heuristics can find negative reduced cost columns, the exact MIP model described in Section 5.1 is used. Similarly, at each iteration, port layouts of negative reduced costs are also generated via a heuristic algorithm (see Section 6.4.3); the most negative reduced cost port layout found (if any) is added to the RMP; if the heuristic cannot find any negative reduced cost port layout, the exact MIP of Section 5.2 is solved. To speed up the solution process, the RMP is initialized with an additional set of columns to stabilize the dual variables (see Section 6.3), and the CG process is terminated as soon as z_{RMP} is not greater than a combinatorial lower bound computed as described in Section 6.2.

The procedure to initialize the RMP, to compute the combinatorial lower bound, the columns added to stabilize the CG procedure, and the three heuristic pricing algorithms are described next.

6.1. Initializing the Restricted Master Problem

The RMP is initialized with the variables $\mathbf{x}$ and $\mathbf{y}$ of the set of stack plans and port layouts of a CSPP solution computed via a greedy heuristic algorithm with a polynomial running time.

The greedy algorithm can be outlined as follows. First, the algorithm decides where each container originating from Port 1 is stowed upon leaving Port 1; this is done in a way that no overstowage occurs. Then, the algorithm iteratively defines the loading/unloading operations in Ports 2 to $n-1$. In each port j, first all j-containers, along with containers stowed on top of them (i.e., containers to shift), are unloaded. Then, all containers either originating from port j or that need to be shifted are loaded on board by starting from the containers destined to the farthest port; in this loading phase, the algorithms makes an attempt to load j-containers on top of other j-containers (if possible) or to fill up entire stacks with containers having the same

destination; when this is not possible, overstowage takes place.

Our computational experiments with the lower bounding procedure suggested that the procedure used to initialize the RMP does not affect the performance of the lower bounding procedure. Therefore, we do not report further details on the greedy heuristic algorithm.

6.2. A Combinatorial Lower Bound

In this section, we describe a simple combinatorial lower bound, LB_0, to the optimal value of (1)–(5). In the lower bounding procedure, CG is stopped as soon as z_{RMP} is not greater than LB_0. This allows to save some CG iterations and speed up the solution process.

Let us call a port $j \in P_3^{n-1}$ a *critical port* if $t_{1k} = 0$ for all $k \in P_{j+1}^n$ and $t_{2k} = 0$ for all $k \in P_3^j$, that is, the first two rows of the transportation matrix $\mathbf{T}$ read as

$$\mathbf{T} = \begin{bmatrix} 0 & t_{12} & t_{13} & \dots & t_{1j} & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & \dots & 0 & t_{2,j+1} & t_{2,j+2} & \dots & t_{2,n} \\ 0 & 0 & 0 & \ddots & & & & & \end{bmatrix}.$$

Let j^* (if any) be the first critical port.

Proposition 1. *If $n \geq 4$, $\bar{r} \geq 2$, $\mathbf{T}$ is full-loading, j^* exists, and $(\sum_{k=3}^{j^*} t_{1k}) \bmod \bar{r} \neq 0$, then at least $\mathrm{LB}_{\mathrm{Shifts}}$ shifts are needed in any feasible stowage plan, where*

$$\mathrm{LB}_{\mathrm{Shifts}} = \min\left\{\left(\sum_{k=3}^{j^*} t_{1k}\right) \bmod \bar{r},\ \bar{r} - \left(\left(\sum_{k=3}^{j^*} t_{1k}\right) \bmod \bar{r}\right)\right\}. \tag{19}$$

Proof. For the sake of clarity, we refer to Figure 4, where white containers represent containers from Port 1 to Port 2, gray containers originate from Port 1 and are destined to Ports $3, 4, \dots, j^*$, and black containers originate from Port 2. Note that, because $((\sum_{k=3}^{j^*} t_{1k}) \bmod \bar{r}) \neq 0$, there has to be at least one stack with both white and gray containers upon leaving Port 1 and at least one stack with both gray and black containers upon leaving Port 2. There are two cases to consider:

Case 1. $\bar{r} - ((\sum_{k=3}^{j^*} t_{1k}) \bmod \bar{r}) \leq (\sum_{k=3}^{j^*} t_{1k}) \bmod \bar{r}$ (i.e., $((\sum_{k=3}^{j^*} t_{1k}) \bmod \bar{r}) \geq \bar{r}/2$)—as all black containers to load in Port 2 are discharged after all gray containers are already on board, there will be at least $\bar{r} - ((\sum_{k=3}^{j^*} t_{1k}) \bmod \bar{r})$ black containers on top of gray containers; this will cause $\bar{r} - ((\sum_{k=3}^{j^*} t_{1k}) \bmod \bar{r})$ shifts in a later port; on the other hand, it would take $(\sum_{k=3}^{j^*} t_{1k}) \bmod \bar{r}$ shifts to avoid overstowage at Port 2.

Case 2. $(\sum_{k=3}^{j^*} t_{1k}) \bmod \bar{r} < \bar{r} - ((\sum_{k=3}^{j^*} t_{1k}) \bmod \bar{r})$ (i.e., $(\sum_{k=3}^{j^*} t_{1k}) \bmod \bar{r} < \bar{r}/2$)—overstowage can be avoided at Port 2 if the $(\sum_{k=3}^{j^*} t_{1k}) \bmod \bar{r}$ gray containers stowed in the stack with white containers are shifted; this implies already $(\sum_{k=3}^{j^*} t_{1k}) \bmod \bar{r}$ shifts; on the other hand, it would not be convenient leaving the gray containers under the black ones because this would cause $\bar{r} - ((\sum_{k=3}^{j^*} t_{1k}) \bmod \bar{r})$ shifts in a later port.

Therefore, at least $\mathrm{LB}_{\mathrm{Shifts}}$ shifts as defined in (19) are necessary. □

Because of Proposition 1, a valid lower bound, LB_0, to the optimal solution of (1)–(5) is $\mathrm{LB}_0 = n\,\mathrm{Cont} + \mathrm{LB}_{\mathrm{Shifts}}$.

6.3. Stabilizing the Master Problem

To limit the computing time of the CG bounding procedure, some stabilization of the dual variables is needed because of high degeneracy of the MP. A successful stabilization technique used in the literature is to add cuts to the MD to limit the range of the dual variables. Ideally, good inequalities restrict the set of feasible solutions of the MD without cutting off any of its optimal solutions so that the final lower bound does not change: such inequalities are called *dual optimal inequalities* (DOIs; Ben Amor, Desrosiers, and Valerio de Carvalho 2006). A category of problems on which DOIs have been extensively studied and successfully applied is cutting stock problems (see, e.g., Ben Amor, Desrosiers, and Valerio de Carvalho 2006; Alves and de Carvalho 2008; Clautiaux et al. 2011).

When we initialize the RMP, we add three sets of inequalities to the MD for each pair of ports $(i, j) \in \hat{\mathcal{T}}$:

- Let $\mathcal{Q}_1 = \{(q_1, q_2): 1 \leq q_1 < q_2 \leq \bar{r} \wedge \hat{t}_{ij} \bmod q_1 = 0 \wedge \hat{t}_{ij} \bmod q_2 = 0\}$, then

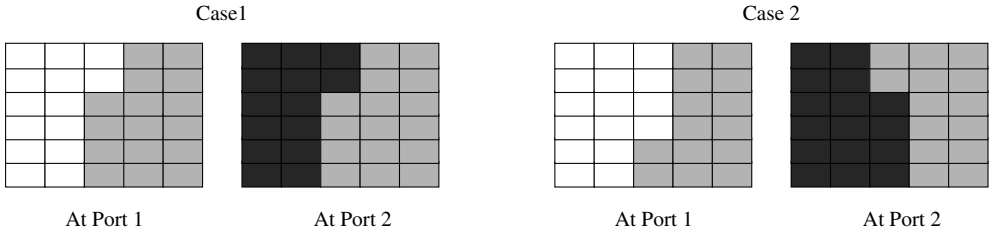
$$v_{ijq_2} \leq \frac{q_2}{q_1} v_{ijq_1}, \quad (q_1, q_2) \in \mathcal{Q}_1. \tag{20}$$

- Let $\mathcal{Q}_2 = \{(q_1, q_2, m): 1 \leq q_1 \leq \bar{r} - 2 \wedge q_1 + 2 \leq q_2 \leq \bar{r} \wedge 2 \leq m < q_2 \wedge (q_1 + q_2) \bmod m = 0\}$, then

$$v_{ijq_1} + v_{ijq_2} \leq m \cdot v_{ijq_3}, \quad (q_1, q_2, m) \in \mathcal{Q}_2, \tag{21}$$

where $q_3 = (q_1 + q_2)/m$.

Figure 4. The Two Cases Considered in the Proof of Proposition 1



- Let $\mathcal{Q}_3 = \{(q_1, q_2, q_3)\colon 1 \le q_1 < q_2 < q_3 < \bar{r} \wedge q_3 = q_1 + q_2\}$, then

$$v_{ijq_3} \le v_{ijq_1} + v_{ijq_2}, \quad (q_1, q_2, q_3) \in \mathcal{Q}_3. \tag{22}$$

The main idea behind inequalities (20)–(22) is to rank the dual variables v_{ijq} in a way that, when solving the pricing problems, columns (both stack plans and port layouts) where containers of the same transport $(i, j) \in \hat{\mathcal{T}}$ are stowed in as few stacks as possible are prioritized over columns where each stack contains containers with many different destinations. Indeed, we have observed that, in most optimal CSPP solutions, there are many stacks with containers with the same destination upon leaving each port and very few stacks are shared by containers with different destinations. Consider the following three examples of inequalities (20)–(22):

- Example of (20). Given $\hat{t}_{ij} = 30$, $q_1 = 1$, and $q_2 = 3$, inequality $v_{ij3} \le 3v_{ij1}$ is added; this prioritizes a port layout where three j-containers are stowed in a single stack upon leaving port i over a port layout where the same three j-containers are stowed in three different stacks.
- Example of (21). Given $q_1 = 1$, $q_2 = 5$, and $m = 3$, inequality $v_{ij1} + v_{ij5} \le 3v_{ij2}$ is added; this prioritizes a port layout where six j-containers are split into two stacks (one with a single j-container and another with five containers) over a port layout where six j-containers are stowed in three different stacks (two containers per stack).
- Example of (22). $q_1 = 2$, $q_2 = 4$, and $q_3 = 6$, then inequality $v_{ij6} \le v_{ij2} + v_{ij4}$ is added; a port layout where six j-containers are stowed in the same stack upon leaving port i is prioritized over a port layout where the six j-containers are stowed in two different stacks (two in one stack and four in the other).

Unfortunately, we did not manage to prove that inequalities (20)–(22) are DOIs, but the computational results reported in Section 8 show that the final lower bound achieved with and without these cuts did not change in the instances we tested, thus suggesting that (20)–(22) may be DOIs. Besides that, adding these cuts to the initial RMP (as extra columns) significantly decreases the total computing time of the bounding procedure—up to 99% in some test instances.

Note that, if inequalities (20)–(22) are not DOIs (and the MP is then overstabilized), the lower bounding procedure is still correct (as we do not require the primal feasibility of the final MP solution), but the final lower bound is simply worse, i.e., strictly lower than z_{MP}.

6.4. Heuristic Algorithms for Pricing Problems

In this section, we describe the three heuristic procedures to price out columns. The first two procedures, *HeuX1* and *HeuX2*, generate heuristic solutions of the pricing problem for stack plans; both procedures are based on *dynamic programming* (DP), but *HeuX1* considers stack plans without any shifts only, while *HeuX2* allows shifts under particular circumstances. The third heuristic algorithm, *HeuY*, prices out port layouts.

6.4.1. Procedure *HeuX1*.

In Section 5.2, we introduced the concept of stack patterns, which is also used in Procedure *HeuX1*. For each stack pattern $h \in \mathcal{H}_i$ of port $i \in P_1^{n-1}$, let us define a set of predecessors $\Gamma(h, i) \subseteq \mathcal{H}_{i-1}$ that contains the subset of stack patterns of port $i-1$ that can turn into stack pattern h after all discharges and loadings are completed, but without shifts. For each stack pattern $h \in \mathcal{H}_1$, the set of predecessors $\Gamma(h, 1)$ is empty; for stack pattern $h \in \mathcal{H}_i$ of port $i \in P_2^{n-1}$, the set $\Gamma(h, i)$ is defined as

$$\Gamma(h,i) = \Big\{h' \in \mathcal{H}_{i-1}\colon (\tau(h') = 0) \vee \big(\tau(h') > 0 \wedge w_j^h = w_j^{h'}, \\ \forall\, j > \tau(h') \wedge w^h_{\tau(h')} \ge w^{h'}_{\tau(h')}\big)\Big\},$$

where, for each $h \in \mathcal{H}_i$, $i \in P_1^{n-2}$, $\tau(h)$ is the first destination port of the containers stowed in stack pattern h, excluding port $i+1$, i.e.,

$$\tau(h) = \begin{cases} 0 & \text{if } w_{i+1}^h = \bar{r}, \\ \underset{j=i+2,\dots,n}{\arg\min}\{w_j^h\colon w_j^h > 0\} & \text{otherwise.} \end{cases}$$

An example of the set $\Gamma(h, i)$ of a stack pattern is provided in Figure 5.

Let $f(h, i)$ be the minimum reduced cost of any stack plan that (a) implements stack pattern $h \in \mathcal{H}_i$ in port $i \in P_1^{n-1}$, (b) has been derived from any stack patterns of the sets $\mathcal{H}_{i'}$ at ports $i' = 1, \dots, i-1$, and (c) does not require any shifts in any port. Moreover, let $v(h)$ be the sum of the dual variables of constraints (3) associated with stack pattern h, defined as $v(h) = \sum_{j \in P_{i+1}^n\colon w_j^h > 0} v_{ij} w_j^h$. The most negative reduced cost stack plan without shifts can be computed with the following DPrecursion:

$$\text{Initialization: } f(h,1) = -v(h), \quad \forall\, h \in \mathcal{H}_1,$$

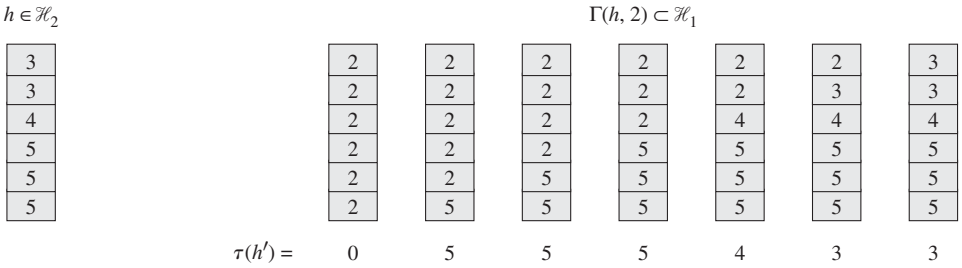
$$\text{Recursive step: } f(h,i) = \min_{h' \in \Gamma(h,i)} \{f(h', i-1) - v(h) + w_i^{h'}\}, \\ \forall\, i \in P_2^{n-1}\ \forall\, h \in \mathcal{H}_i.$$

Note that $\mathcal{H}_{n-1}$ contains one vector only (i.e., $\mathbf{w}^h = (w_n^h) = (\bar{r})$). The stack plan of minimum reduced cost is given by the stack plan $h \in \mathcal{H}_{n-1}$ and has a reduced cost $f(h, n-1) + \bar{r}$. If it has a negative reduced cost, it is added to the RMP whenever procedure *HeuX1* is called.

6.4.2. Procedure *HeuX2*.

Let us define a *stack layout* of port $i \in P_1^{n-1}$ as a vector $\boldsymbol{\lambda} \in \{0,1\}^{(n-i)\bar{r}}$, where λ_{jr} ($j \in P_{i+1}^n$, $r \in R$) is equal to 1 if a j-container is stowed in tier r of a given stack upon leaving port i (0 otherwise). Overstowage is allowed in stack layouts. Note that a stack plan consists of a stack layout for each port $i \in P_1^{n-1}$. Among all possible stack layouts, let us consider the following two sets of stack layouts for port $i \in P_1^{n-1}$:

Figure 5. An Example of Stack Pattern $\mathbf{w}^h = (w_3^h, w_4^h, w_5^h) = (2,1,3)$ for Port 2 (Left Panel) and Its Predecessors (Right Panel) for an Instance with Five Ports and Six Tiers per Stack



- Stack layouts without overstowage, with containers with at most four different destinations, and with those destined to the same port stowed in consecutive tiers. Given an additional auxiliary variable $\eta_j \in \{0,1\}$ ($j \in P_{i+1}^n$) indicating if at least one j-container is stowed (0 otherwise), this set of stack layouts corresponds to all feasible solutions $\boldsymbol{\lambda} \in \{0,1\}^{(n-i)\bar{r}}$ of the following constraints:

$$\sum_{j=i+1}^{n} \lambda_{jr} = 1, \quad r \in R, \tag{23}$$

$$\sum_{r \in R} \lambda_{jr} \le \min\{\bar{r}, \hat{t}_{ij}\}\eta_j, \quad j \in P_{i+1}^n, \tag{24}$$

$$\sum_{j=i+1}^{n} \eta_j \le 4, \tag{25}$$

$$\sum_{j=i+1}^{n} j\lambda_{jr} \ge \sum_{j=i+1}^{n} j\lambda_{j,r+1}, \quad r \in R^-, \tag{26}$$

where constraints (23) state that exactly one container has to be stowed in each tier, constraints (24) are on/off constraints to set η_j equal to 1 whenever at least a j-container is stowed in the stack layout, constraints (25) ensure that containers are destined to no more than four ports, and constraints (26) prevent overstowage.

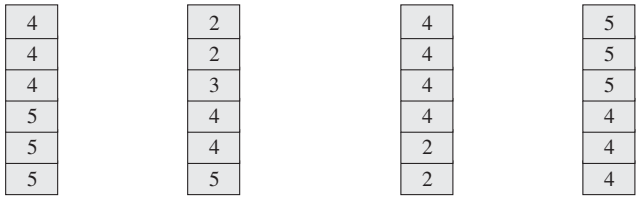
- All stack layouts with containers destined to exactly two ports, with containers destined to the same port stowed in consecutive tiers, and with overstowage. Therefore, this set of stack layouts corresponds to all feasible solutions of the following constraints:

$$\sum_{j=i+1}^{n} \eta_j = 2, \tag{27}$$

$$\sum_{j=i+1}^{n} j\lambda_{jr} \le \sum_{j=i+1}^{n} j\lambda_{j,r+1}, \quad r \in R^-,$$
$$(23)\text{–}(24). \tag{28}$$

For each port $i \in P_1^{n-1}$, let us define the set Λ_i of stack layouts as $\Lambda_i = \{\boldsymbol{\lambda} \in \{0,1\}^{(n-i)\bar{r}}\text{: (23)–(26)}\} \cup \{\boldsymbol{\lambda} \in \{0,1\}^{(n-i)\bar{r}}\text{: (23)–(24), (27)–(28)}\}$. Note that Λ_i contains a subset of all feasible stack layouts that make up the entire set $\mathcal{S}$ of stack plans defined in Section 4. Four exampless of stack layouts of the set Λ_1 of Port 1 for an instance with five ports and six tiers per stack are given in Figure 6; the first two stack layouts on the left satisfy constraints (23)–(26), and the second two stack layouts on the right satisfy constraints (23)–(24) and (27)–(28).

Figure 6. Four Stack Layouts of the Set Λ_1 of Port 1 for a CSPP with Five Ports and Six Tiers per Stack



Let $v(\boldsymbol{\lambda})$ be the sum of the dual variables of (3) associated with stack layout $\boldsymbol{\lambda} \in \Lambda_i$ ($i \in P_1^{n-1}$) defined as $v(\boldsymbol{\lambda}) = \sum_{j \in P_{i+1}^n : w_j > 0} v_{ij} w_j$, where $w_j = \sum_{r \in R} \lambda_{jr}$. Moreover, let $g(\boldsymbol{\lambda}', \boldsymbol{\lambda})$ be a *compare* function that returns the number of unloading operations (if any) that need to be performed to implement stack layout $\boldsymbol{\lambda}'$ at port $i-1$ and stack layout $\boldsymbol{\lambda}$ at port i, with $i \in P_2^{n-1}$, defined as

$$g(\boldsymbol{\lambda}', \boldsymbol{\lambda}) = \begin{cases} 0 & \text{if } \lambda'_{jr} = \lambda_{jr}, \forall j \in P_{i+1}^n, \forall r \in R, \\ \bar{r} - r^*(\boldsymbol{\lambda}', \boldsymbol{\lambda}) + 1 & \text{otherwise,} \end{cases}$$

where $r^*(\boldsymbol{\lambda}', \boldsymbol{\lambda}) = \arg\min_{r \in R}\{\lambda'_{jr} \ne \lambda_{jr} \text{: } j \in P_{i+1}^n\}$, that is, $r^*(\boldsymbol{\lambda}', \boldsymbol{\lambda})$ is the index of the first tier from the bottom of the stack where the container stowed in stack layout $\boldsymbol{\lambda}'$ has a destination port different from the container stowed in the same tier of stack layout $\boldsymbol{\lambda}$.

Let $f(\boldsymbol{\lambda}, i)$ be the minimum reduced cost of any stack plan that implements stack layout $\boldsymbol{\lambda}$ at Port $i \in P_1^{n-1}$, and has implemented any stack layout of the sets Λ_l at Ports $l = 1, \ldots, i-1$. Functions $f(\boldsymbol{\lambda}, i)$ can be computed with the following DP recursion:

Initialization: $f(\boldsymbol{\lambda}, 1) = -v(\boldsymbol{\lambda}) \quad \forall \boldsymbol{\lambda} \in \Lambda_1$,

Recursive step:

$$f(\boldsymbol{\lambda}, i) = \min_{\boldsymbol{\lambda}' \in \Lambda_{i-1}} \{f(\boldsymbol{\lambda}', i-1) - v(\boldsymbol{\lambda}) + g(\boldsymbol{\lambda}', \boldsymbol{\lambda})\}, \quad \forall i \in P_2^{n-1} \ \forall \boldsymbol{\lambda} \in \Lambda_i.$$

Note that Λ_{n-1} contains one vector only (i.e., $\boldsymbol{\lambda} = (\lambda_{nr}) = 1, \forall r \in R$). A stack plan of minimum reduced

cost w.r.t. the sets Λ_i is given by this vector $\boldsymbol{\lambda} \in \Lambda_{n-1}$ and has reduced cost $f(\boldsymbol{\lambda}, n-1) + \bar{r}$.

6.4.3. Procedure *HeuY*. Solving the pricing problem for port layouts to optimality (as described in Section 5.2) at each iteration of the lower bounding procedure is time-consuming. Therefore, we rely on a heuristic pricing algorithm based on the MIP model (15)–(18). In particular, we set equal to zero all χ_{ih} variables corresponding to stack patterns $h \in \mathcal{H}_i$ having containers destined to more than three ports, i.e., $\chi_{ih} = 0$ if $|\{j\colon j \in P_{i+1}^n \wedge w_j^h > 0\}| > 3$, and solve (15)–(18) with the remaining χ_{ih} variables.

7. Finding an Optimal Stowage Plan

The lower bounding procedure described in Section 6 returns a lower bound LB to the CSPP, but does not necessarily provide any (optimal) stowage plan. We could not envision promising ways of embedding the procedure into an exact branch-and-price algorithm, so we propose a solution method, based on the lower bound LB, that chooses between two MIP models that we propose in this section to find an optimal solution.

If LB = n Cont, the MIP model described in Section 7.1 is solved to find a stowage plan with no shifts; clearly, if such a plan exists, it is also optimal for CSPP. Otherwise, if LB > n Cont (i.e., at least LB − n Cont shifts are necessary), another MIP model (see Section 7.2) where shifts are allowed in a small subset of the stacks is solved; if a stowage plan with a number of shifts equal to LB − n Cont is found, then it is clearly an optimal one for CSPP.

Such a solution method strongly relies on the quality of the lower bound LB and has the drawback that no feasible stowage plans may be found. Computational results in Section 8 show that it is still a successful solution method for most test instances with up to 10 ports and 500 stacks.

7.1. Finding an Optimal Stowage Plan Without Shifts

The MIP model described in this section is run if the bounding procedure suggests that a stowage plan with no shifts may exist (i.e., whenever LB = n Cont). Shifts are prevented, so if the optimal CSPP solution requires a strictly positive number of shifts, it cannot be found by this MIP.

Let $\omega_{ijc} \in \mathbb{Z}_+$ be an integer variable equal to the number of j-containers loaded in stack $c \in C$ in port $i \in P_1^{n-1}$ (which is different from the number of j-containers stowed in stack $c \in C$ upon leaving port i), and let $\zeta_{ijc} \in \{0,1\}$ be a binary variable equal to 1 if, upon leaving port $i \in P_1^{n-3}$, in stack $c \in C$ there is at least one container destined to a port $k \in P_{i+2}^{j-1}$, where $j \in P_{i+3}^n$. A stowage plan without shifts corresponds to an optimal solution of the following problem having an optimal value z_0 equal to n Cont:

$$z_0 = \max \sum_{(i,j)\in\mathcal{T}} \sum_{c\in C} \omega_{ijc} \tag{29}$$

$$\text{s.t.} \sum_{c\in C} \omega_{ijc} \le t_{ij}, \quad (i,j) \in \mathcal{T}, \tag{30}$$

$$\sum_{\substack{(k,j)\in\mathcal{T}:\\ k\in P_1^i \wedge j\in P_{i+1}^n}} \omega_{kjc} \le \bar{r}, \quad i \in P_1^{n-1},\, c \in C, \tag{31}$$

$$\sum_{\substack{(h,k)\in\mathcal{T}:\\ h\in P_1^i \wedge k\in P_{i+2}^{j-1}}} \omega_{hkc} \le \bar{r}\zeta_{ijc}, \quad i \in P_1^{n-3},\, j \in P_{i+3}^n,\, c \in C, \tag{32}$$

$$\sum_{(i+1,k)\in\mathcal{T}:\, k\in P_j^n} \omega_{i+1,k,c} \le \bar{r}(1-\zeta_{ijc}), \quad i \in P_1^{n-3},\, j \in P_{i+3}^n,\, c \in C, \tag{33}$$

$$\omega_{ijc} \in \mathbb{Z}_+, \quad (i,j) \in \mathcal{T},\, c \in C, \tag{34}$$

$$\zeta_{ijc} \in \{0,1\}, \quad i \in P_1^{n-3},\, j \in P_{i+3}^n,\, c \in C. \tag{35}$$

The objective function (29) maximizes the number of containers that are loaded on board in each port, which is at most n Cont. Constraints (30) state that each transport $(i,j) \in \mathcal{T}$ must be satisfied. Constraints (31) impose on the solution not to exceed the number of tiers per stack. Constraints (32) are on/off constraints for variables ζ_{ijc} setting them equal to 1, for a given port $i \in P_1^{n-3}$, a given port $j \in P_{i+3}^n$, and a given stack $c \in C$, whenever stack c contains containers destined to one of the ports from $i+2$ to $j-1$. Constraints (33) stipulate that, if ζ_{ijc} is equal to 1, then containers destined to ports $j, j+1, \ldots, n$ cannot be loaded into stack $c \in C$ in port $i+1$. Constraints (34)–(35) define the range of the decision variables.

To better clarify constraints (32) and (33), let us consider the following example: $n = 8$, $\bar{r} = 10$, and, for a given stack $c \in C$, $\omega_{12c} = 6$, $\omega_{17c} = 1$, $\omega_{18c} = 3$, $\omega_{23c} = 4$, $\omega_{27c} = 2$. In stack c at Port 1, three 8-containers have been stowed in the bottommost tiers, one 7-container has been stowed on top of the previous three containers, and six 2-containers have been stowed in the topmost tiers. At Port 2, the six 2-containers have been discharged and have been replaced by two 7-containers and four 3-containers. In Port 3, the four topmost containers will be discharged, but, as overstowage is prevented, containers destined to Port 8 cannot be further loaded in stack c. Indeed, the left-hand side of constraint (32) for $i = 2$ and $j = 8$ reads as $\omega_{14c} + \omega_{15c} + \omega_{16c} + \omega_{17c} + \omega_{24c} + \omega_{25c} + \omega_{26c} + \omega_{27c} = 3$, which implies $\zeta_{28c} = 1$. Constraint (33) for $i = 2$ and $j = 8$ sets ω_{38c} equal to 0, thus preventing 8-containers to be loaded in the stack in Port 3. On the contrary, constraints (32) leave $\zeta_{25c} = \zeta_{26c} = \zeta_{27c} = 0$, so ω_{35c}, ω_{36c}, and ω_{37c} are allowed to take strictly positive values because of constraints (33). Indeed, in Port 3, containers destined to Ports 5, 6, and 7 can replace the four 3-containers that are discharged.

7.2. Finding an Optimal Stowage Plan with Shifts

Whenever the lower bound LB returned by the bounding procedure of Section 6 is strictly greater than n Cont, shifts are necessary in any feasible stowage plan. In this case, to find an optimal stowage plan, we run the MIP model described in this section. This MIP extends model (29)–(35) by allowing shifts in a limited number of stacks only and preventing them in the remaining stacks.

Let $C_N \subset C$ be the subset of stacks where shifts are prevented, and let $C_Y = C \backslash C_N$ be the remaining subset of stacks where shifts are allowed.

Five sets of decision variables are used. The first two sets, ω_{ijc} and ζ_{ijc}, are defined as in Section 7.1 but on the set of stacks C_N, that is, $\omega_{ijc} \in \mathbb{Z}_+$ equals the number of j-containers added in stack $c \in C_N$ at port $i \in P_1^{n-1}$, and $\zeta_{ijc} \in \{0,1\}$ equals 1 if, upon leaving port $i \in P_1^{n-3}$, in stack $c \in C_N$ there is at least one container destined to a port $k \in P_{i+2}^{j-1}$, where $j \in P_{i+3}^{n}$. The other three sets of variables are $\mu_{ijrc} \in \{0,1\}$, which equals 1 if, upon leaving port $i \in P_1^{n-1}$, a j-container (with $j \in P_{i+1}^{n}$) is stowed in tier $r \in R$ of stack $c \in C_Y$ (0 otherwise); $\gamma_{irc} \in \{0,1\}$, which equals 1 if the container stowed in tier $r \in R$ of stack $c \in C_Y$ upon leaving Port $i \in P_1^{n-1}$ is unloaded at Port $i+1$; and $\delta_{ij} \in \mathbb{R}_+$, which indicates the number of j-containers that should be on board upon leaving Port $i \in P_1^{n-1}$ but were not loaded. The proposed MIP model reads as follows:

$$z_1 = \min\left\{ \sum_{(i,j)\in\hat{\mathcal{T}}} M\delta_{ij} + \sum_{(i,j)\in\mathcal{T}} \sum_{c\in C_N} \omega_{ijc} + \sum_{i=1}^{n-1} \sum_{r\in R} \sum_{c\in C_Y} \gamma_{irc} \right\} \quad (36)$$

$$\text{s.t.} \sum_{c\in C_N} \omega_{ijc} + \sum_{c\in C_Y} \sum_{r\in R} (\mu_{ijrc} - \mu_{i-1,jrc}) \le t_{ij}, \quad (i,j)\in\mathcal{T}, \quad (37)$$

$$\sum_{\substack{(k,j)\in\mathcal{T}:\\ k\in P_1^i \wedge j\in P_{i+1}^n}} \omega_{kjc} \le \bar{r}, \quad i\in P_1^{n-1}, c\in C_N, \quad (38)$$

$$\sum_{\substack{(h,k)\in\mathcal{T}:\\ h\in P_1^i \wedge k\in P_{i+2}^{j-1}}} \omega_{hkc} \le \bar{r}\zeta_{ijc}, \quad i\in P_1^{n-3}, j\in P_{i+3}^{n}, c\in C_N, \quad (39)$$

$$\sum_{(i+1,k)\in\mathcal{T}: k\in P_j^n} \omega_{i+1,kc} \le \bar{r}(1-\zeta_{ijc}), \quad i\in P_1^{n-3}, j\in P_{i+3}^{n}, c\in C_N, \quad (40)$$

$$\sum_{r\in R} \sum_{c\in C_Y} \mu_{ijrc} + \sum_{\substack{(k,j)\in\mathcal{T}: c\in C_N\\ k\in P_1^i}} \omega_{kjc} + \delta_{ij} \ge \hat{t}_{ij}, \quad (i,j)\in\hat{\mathcal{T}}, \quad (41)$$

$$\sum_{j=i+1}^{n} \mu_{ijrc} \le 1, \quad i\in P_1^{n-1}, r\in R, c\in C_Y, \quad (42)$$

$$\gamma_{irc} \ge \mu_{ijrc} - \mu_{i+1,jrc}, \quad (i,j)\in\hat{\mathcal{T}}, r\in R, c\in C_Y, \quad (43)$$

$$\gamma_{irc} \le \gamma_{i,r+1,c}, \quad i\in P_1^{n-1}, r\in R^-, c\in C_Y, \quad (44)$$

$$\omega_{ijc} \in \mathbb{Z}_+, \quad (i,j)\in\mathcal{T}, c\in C_N, \quad (45)$$

$$\zeta_{ijc} \in \{0,1\}, \quad i\in P_1^{n-3}, j\in P_{i+3}^{n}, c\in C_N, \quad (46)$$

$$\mu_{ijrc} \in \{0,1\}, \quad (i,j)\in\hat{\mathcal{T}}, r\in R, c\in C_Y, \quad (47)$$

$$\gamma_{irc} \in \{0,1\}, \quad i\in P_1^{n-1}, r\in R, c\in C_Y, \quad (48)$$

$$\delta_{ij} \in \mathbb{R}_+, \quad (i,j)\in\mathcal{T}, \quad (49)$$

where M in the objective function (36) represents a large enough number.

The objective function (36), first, minimizes the number of containers not loaded (i.e., aims to set all $\boldsymbol{\delta}$ variables to 0) and then minimizes the total number of shifts; note that any feasible stowage plan corresponds to a solution where all δ_{ij} are 0, and the second and third term of the objective function count the number of discharges in stacks C_N and the number of discharges plus the number of shifts in stacks C_Y, respectively. Constraints (37) guarantee that, if all containers are loaded, each transport t_{ij}, $(i,j) \in \mathcal{T}$, is not exceeded. Constraints (38)–(40) correspond to constraints (31)–(33). Constraints (41) link variables ω_{ijc}, μ_{ijrc}, and δ_{ij} and set δ_{ij} equal to the number of j-containers not loaded upon leaving Port i (with $(i,j) \in \hat{\mathcal{T}}$). Constraints (42) stipulate that at most one container can be stowed in each tier $r \in R$ of each stack $c \in C_Y$ upon leaving port $i \in P_1^{n-1}$. Constraints (43) are on-off constraints for variables γ_{irc} that are set equal to 1 whenever the container stowed in tier $r \in R$ of stack $c \in C_Y$ upon leaving Port $i \in P_1^{n-1}$ is unloaded at Port $i+1$. Constraints (44) stipulate that the container loaded in tier $r+1$ of stack $c \in C_Y$ at port $i \in P_1^{n-1}$ has to be unloaded if the container stowed in tier $r \in R^-$ (or any of the tiers underneath) is unloaded. Constraints (45)–(49) define the range of the decision variables.

To make use of the lower bound LB in model (36)–(49), the following constraint is added:

$$\sum_{(i,j)\in\mathcal{T}} \sum_{c\in C_N} \omega_{ijc} + \sum_{i=1}^{n-1} \sum_{r\in R} \sum_{c\in C_Y} \gamma_{irc} \ge \text{LB}, \quad (50)$$

which allows to stop the MIP model as soon as a solution of cost LB is found. Moreover, the number of stacks where shifts are allowed is set equal to the minimum number of shifts returned by LB, that is, $|C_Y| = \min\{\bar{c}, \text{LB} - n \text{ Cont}\}$.

8. Computational Results

In this section, we report the computational results of the lower bounding procedure described in Section 6 and the proposed method for finding an optimal stowage plan described in Section 7. A comparison with the compact formulations of Avriel and Penn (1993) and Ding and Chou (2015), which represent the state-of-the-art exact methods to solve the CSPP, is provided; both formulations are reported in the online appendix. We coded both formulations and solved them by using Cplex (version 12.6.1). Our algorithms were coded in C compiled with Visual Studio 2015 and by using Cplex callable libraries (version 12.6.1) to solve formulation (6)–(14) to price out stack plans, formulation (15)–(18) to price out port layouts, and formulations (29)–(35), (36)–(49) to find an optimal stowage plan. All algorithms and models were run on a single

thread of an Intel Core i7-6700K at 4 GHz equipped with 24 GB RAM. All computing times are in seconds rounded to the nearest integer.

8.1. Test Instances

As test instances used in prior papers were not available, we generated 405 instances of five types (`Long`, `Mixed`, `Short`, `Authentic`, and `Required`) with different features of the transportation matrix. All transportation matrices are full-loading as assumed throughout the paper.

`Long`, `Mixed`, and `Short` instances were described and tested by Avriel et al. (1998). In `Long` instances, containers stay on average a long period of time on board. In `Short` instances, containers stay on average a short period of time on board. In `Mixed` instances, there is a more random distribution of the magnitude of the transports, so long and short distance transports are mixed. The reader is referred to Avriel et al. (1998) for a detailed description of these three sets.

`Authentic` instances were described in Ding and Chou (2015) and combine both short and long distance transports. We coded the instance generator that is accurately described in Ding and Chou (2015), so we omit the details here.

The last set of instances, `Required`, is introduced in this paper. Transportation matrices are such that shifts are necessary, that is $LB_{Shifts} > 0$ (see (19)). To do so, the first rows of $\mathbf{T}$ (a random number typically between two and four) contain a critical port (see Section 6.2), thus guaranteeing that shifts must be performed. In each row of $\mathbf{T}$, the nonzero transports represent partitions of the $\bar{c} \cdot \bar{r}$ containers to transport to fulfill the full-loading assumption. For the sake of brevity, we omit further details on the instance generator, whose source code is available upon request to the authors.

For each type, we generated 81 instances by varying the number of ports ($n = 6, 8, 10$), the number of stacks ($\bar{c} = 100, 300, 500$), and the number of tiers per stack ($\bar{r} = 6, 8, 10$). For each combination of n, $\bar{c}$, and $\bar{r}$, three instances were generated. The parameters n, $\bar{c}$, and $\bar{r}$ we selected correspond to real-life problems of small/medium-size container ships transporting 600 to 5,000 TEU and are in line with what has been done in all of the papers from the literature, except for the number of ports: heuristic methods can address instances with n up to 20, which is out of reach for our solution method that aims to find an optimal stowage plan.

8.2. Performance Analysis

Tables 2–6 report a summary of the computation results achieved by the compact formulation of Avriel and Penn (1993) (hereafter called AP93), the compact formulation of Ding and Chou (2015) (hereafter DC15), and

Table 2. Summary of the Computational Results on `Long` Instances

			AP93					DC15					This paper							
n	$\bar{r}$	$\bar{c}$	LB	UB	Opt	T_{ub}	T_{tot}	LB	UB	Opt	T_{ub}	T_{tot}	LB_0	Vars	Iter	LB	T_{lb}	UB	Opt	T_{tot}
6	6	100	0	0	3	1	1	0	0	3	11	11	0	305	4	0	0	0	3	0
6	6	300	0	0	3	4	4	0	0	3	74	74	0	506	6	0	0	0	3	0
6	6	500	0	0	3	4	4	0	0	3	60	60	0	659	3	0	0	0	3	0
6	8	100	0	0	3	2	2	0	0	3	11	11	0	519	6	0	0	0	3	0
6	8	300	0	0	3	9	9	0	0	3	482	482	0	722	5	0	0	0	3	0
6	8	500	0	0	3	33	33	0	0	3	1,666	1,666	0	911	4	0	0	0	3	1
6	10	100	0	0	3	4	4	0	0	3	54	54	0	761	12	0	0	0	3	0
6	10	300	0	0	3	9	9	0	0	3	232	232	0	827	3	0	0	0	3	0
6	10	500	0	0	3	462	462	0	8	2	1,994	2,061	0	1,159	4	0	0	0	3	0
8	6	100	0	0	3	8	8	0	0	3	134	134	0	471	19	0	0	0	3	0
8	6	300	0	0	3	57	57	0	0	3	1,389	1,389	0	677	8	0	0	0	3	4
8	6	500	0	0	3	95	95	0	19	1	2,685	3,088	0	868	8	0	0	0	3	7
8	8	100	0	0	3	19	19	0	0	3	471	471	0	877	20	0	0	0	3	0
8	8	300	0	0	3	64	64	0	0	3	2,212	2,212	0	1,003	8	0	0	0	3	1
8	8	500	0	0	3	321	321	0	14,751	0	807	3,600	0	1,231	7	0	0	0	3	4
8	10	100	0	0	3	41	41	0	0	3	994	994	0	1,324	14	0	0	0	3	1
8	10	300	0	0	3	240	240	0	3,831	1	1,551	2,871	0	1,503	11	0	0	0	3	2
8	10	500	0	0	3	495	495	0	18,779	0	1,758	3,600	0	1,763	10	0	0	0	3	5
10	6	100	0	0	3	6	6	0	0	3	192	192	0	604	19	0	0	0	3	0
10	6	300	0	0	3	204	204	0	126	0	2,291	3,600	0	877	13	0	0	0	3	7
10	6	500	0	0	3	852	852	0	15,959	0	1,826	3,600	0	1,110	19	0	0	0	3	13
10	8	100	0	0	3	92	92	0	3	2	814	1,923	0	1,149	27	0	0	0	3	0
10	8	300	0	0	3	467	467	0	7,701	1	1,332	2,736	0	1,452	19	0	0	0	3	10
10	8	500	0	0	3	1,505	1,505	0	20,051	0	3,312	3,600	0	1,620	14	0	0	0	3	37
10	10	100	0	0	3	264	264	0	58	1	2,624	2,795	0	1,911	26	0	0	0	3	2
10	10	300	0	0	3	862	862	0	16,093	0	1,802	3,600	0	2,228	20	0	0	0	3	13
10	10	500	0	8,268	2	1,293	2,412	0	30,342	0	3,600	3,600	0	2,320	19	0	0	0	3	17
Avg						274	316				1,273	1,802		1,087	12		0			5
Solved					80					50									81	

Table 3. Summary of the Computational Results on `Mixed` Instances

			AP93					DC15					This paper							
n	$\bar{r}$	$\bar{c}$	LB	UB	Opt	T_{ub}	T_{tot}	LB	UB	Opt	T_{ub}	T_{tot}	LB_0	Vars	Iter	LB	T_{lb}	UB	Opt	T_{tot}
6	6	100	0	0	3	1	1	0	0	3	5	5	0	324	4	0	0	0	3	0
6	6	300	0	0	3	4	4	0	0	3	88	88	0	511	5	0	0	0	3	0
6	6	500	0	0	3	12	12	0	0	3	825	825	0	701	4	0	0	0	3	0
6	8	100	0	0	3	2	2	0	0	3	30	30	0	522	3	0	0	0	3	0
6	8	300	0	0	3	10	10	0	0	3	230	230	0	719	4	0	0	0	3	0
6	8	500	0	0	3	28	28	0	0	3	1,909	1,909	0	912	3	0	0	0	3	1
6	10	100	0	0	3	4	4	0	0	3	105	105	0	796	3	0	0	0	3	0
6	10	300	0	0	3	21	21	0	0	3	626	626	0	964	3	0	0	0	3	0
6	10	500	0	0	3	62	62	0	0	3	2,817	2,817	0	1,207	3	0	0	0	3	1
8	6	100	0	0	3	6	6	0	0	3	123	123	0	481	11	0	0	0	3	0
8	6	300	0	0	3	32	32	0	0	3	1,603	1,603	0	671	5	0	0	0	3	2
8	6	500	0	0	3	96	96	0	3,533	1	1,699	3,307	0	871	9	0	0	0	3	4
8	8	100	0	0	3	21	21	0	0	3	354	354	0	876	10	0	0	0	3	1
8	8	300	0	0	3	84	84	0	4	2	2,566	2,957	0	1,042	8	0	0	0	3	2
8	8	500	0	0	3	171	171	0	8,625	0	1,322	3,600	0	1,226	6	0	0	0	3	4
8	10	100	0	0	3	46	46	0	0	3	847	847	0	1,387	9	0	0	0	3	0
8	10	300	0	0	3	241	241	0	3,582	1	2,281	3,270	0	1,516	9	0	0	0	3	2
8	10	500	0	0	3	451	451	0	15,435	0	2,145	3,600	0	1,839	7	0	0	0	3	4
10	6	100	0	0	3	18	18	0	0	3	1,440	1,440	0	672	14	0	0	0	3	1
10	6	300	0	0	3	138	138	0	2,769	1	2,011	3,553	0	854	13	0	0	0	3	7
10	6	500	0	0	3	461	461	0	10,151	0	2,300	3,600	0	1,086	15	0	0	0	3	35
10	8	100	0	0	3	53	53	0	5	2	1,694	2,040	0	1,276	16	0	0	0	3	1
10	8	300	0	0	3	195	195	0	6,564	0	1,540	3,600	0	1,394	13	0	0	0	3	8
10	8	500	0	0	3	852	852	0	16,688	0	2,703	3,600	0	1,692	14	0	0	0	3	26
10	10	100	0	0	3	148	148	0	166	1	2,091	3,008	0	1,911	19	0	0	0	3	1
10	10	300	0	0	3	996	996	0	14,859	0	1,414	3,600	0	2,178	18	0	0	0	3	9
10	10	500	0	0	3	1,869	1,869	0	24,247	0	3,292	3,600	0	2,461	15	0	1	0	3	19
Avg						223	223				1,410	2,012		1,114	9		0			5
Solved					81					50									81	

the solution method proposed in this paper. Each table refers to a different type of instance and each line indicates averages over the three instances with fixed values n, $\bar{c}$, and $\bar{r}$. A time limit of one hour was imposed on each solution method. Detailed computational results are reported in the online appendix.

The following information is reported in each table: features of the instance (n, $\bar{c}$, $\bar{r}$); for AP93 and for DC15, the final lower bound (LB), the final upper bound (UB), the number of instances solved to optimality out of the three (Opt), the computing time (T_{ub}) to find the final upper bound (UB), and the total computing time (T_{tot}); for our solution method, the combinatorial lower bound (LB_0), the number of variables of the final RMP (Vars), the number of iterations of column generation (Iter), the lower bound returned by the bounding procedure (LB), the computing time taken by the bounding procedure (T_{lb}), the final upper bound (UB), the number of instances solved to optimality (Opt), and the total computing time (T_{tot}). In LB_0, LB, and UB of our solution method, we subtracted n Cont (i.e., the constant number of discharges) to have a direct comparison with AP93 and DC15 (the reader is referred to the online appendix for a detailed description of AP93 and DC15); moreover, in two instances no feasible UB was found, so the reported UB is the average of the costs of the feasible solutions only—in this case, an asterisk is also reported. The last two lines of each table report the number of instances solved by each method and average computing times (in which 3,600 seconds are considered whenever the time limit was reached).

Table 2 refers to `Long` instances (detailed results can be found in Tables EC.1–EC.3 in the online appendix). Our solution method was able to find an optimal solution with no shifts for all 81 instances, so all lower bounds are obviously zero. The average computing time of our method is five seconds, so it clearly outperforms both AP93, which could not solve one of the instances with 500 stacks and solved the remaining ones on average in 316 seconds, and DC15, which solved only 50 instances in 1,802 seconds of average computing time. Note that the average computing time of the proposed lower bounding procedure is almost negligible (less than a second), just a few iterations of column generation are needed (12 on average), and the final RMP contains 1,087 variables on average.

Table 3 summarizes the results on `Mixed` instances (see Tables EC.4–EC.6 in the online appendix for detailed results). Our solution method proved that a solution with no shifts can be found on all 81 instances in an average computing time of five seconds. Our method outperformed AP93, which could solve all instances but

Table 4. Summary of the Computational Results on `Short` Instances

			AP93					DC15					This paper							
n	$\bar{r}$	$\bar{c}$	LB	UB	Opt	T_{ub}	T_{tot}	LB	UB	Opt	T_{ub}	T_{tot}	LB_0	Vars	Iter	LB	T_{lb}	UB	Opt	T_{tot}
6	6	100	0	0	3	1	1	0	0	3	4	4	0	305	7	0	0	0	3	0
6	6	300	0	0	3	4	4	0	0	3	29	29	0	503	5	0	0	0	3	0
6	6	500	0	0	3	7	7	0	0	3	72	72	0	707	4	0	0	0	3	1
6	8	100	0	0	3	2	2	0	0	3	8	8	0	513	6	0	0	0	3	0
6	8	300	0	0	3	7	7	0	0	3	38	38	0	709	6	0	0	0	3	0
6	8	500	0	0	3	31	31	0	0	3	1,278	1,278	0	908	3	0	0	0	3	0
6	10	100	0	0	3	5	5	0	0	3	51	51	0	789	6	0	0	0	3	0
6	10	300	0	0	3	19	19	0	0	3	413	413	0	975	4	0	0	0	3	0
6	10	500	0	0	3	65	65	0	0	3	2,212	2,212	0	1,197	3	0	0	0	3	0
8	6	100	0	0	3	8	8	0	0	3	93	93	0	496	13	0	0	0	3	0
8	6	300	0	0	3	46	46	0	0	3	662	662	0	692	9	0	0	0	3	2
8	6	500	0	0	3	83	83	0	83	1	1,216	2,551	0	912	7	0	0	0	3	4
8	8	100	0	0	3	14	14	0	0	3	479	479	0	888	14	0	0	0	3	0
8	8	300	0	0	3	68	68	0	0	3	2,700	2,700	0	1,068	10	0	0	0	3	2
8	8	500	0	0	3	211	211	0	7,423	0	848	3,600	0	1,297	9	0	0	0	3	4
8	10	100	0	0	3	19	19	0	0	3	560	560	0	1,319	12	0	0	0	3	0
8	10	300	0	0	3	195	195	0	2,975	0	1,282	3,600	0	1,633	10	0	0	0	3	2
8	10	500	0	0	3	463	463	0	12,558	0	1,538	3,600	0	1,815	6	0	0	0	3	4
10	6	100	0	0	3	34	34	0	11	2	1,028	1,349	0	697	18	0	0	0	3	1
10	6	300	0	0	3	226	226	0	25	1	2,106	3,484	0	961	15	0	0	0	3	6
10	6	500	0	0	3	317	317	0	6,514	0	1,744	3,600	0	1,129	13	0	0	0	3	18
10	8	100	0	0	3	49	49	0	1	2	2,744	2,986	0	1,330	22	0	0	0	3	1
10	8	300	0	0	3	868	868	0	8,658	0	897	3,600	0	1,590	16	0	0	0	3	8
10	8	500	0	0	3	2,270	2,270	0	14,603	0	2,930	3,600	0	1,824	16	0	0	0	3	23
10	10	100	0	0	3	244	244	0	156	2	2,503	2,658	0	2,056	22	0	0	0	3	1
10	10	300	0	0	3	959	959	0	10,190	0	1,230	3,600	0	2,342	16	0	0	0	3	10
10	10	500	0	6,115	2	556	1,668	0	16,796	0	3,318	3,600	0	2,609	16	0	1	0	3	16
Avg						251	292				1,185	1,868		1,158	11		0			4
Solved					80					50									81	

with an average computing time of 223 seconds, and DC15, which could solve only 50 of the 81 instances. Also, in these instances, the average computing time of the bounding procedure is less than a second and just a few iterations of column generation are performed.

In Table 4, we compare the three solution methods on `Short` instances (see Tables EC.7–EC.9 in the online appendix for detailed results). The results confirm what we have already seen in Tables 2 and 3: our solution method found an optimal solution with no shifts in all 81 instances and outperformed both AP93 and DC93 in terms of number of instances solved to optimality and average computing times.

Table 5 summarizes the results on `Authentic` instances (refer to Tables EC.10–EC.12 in the online appendix for detailed results). Even on these instances, no shifts were necessary in any of the optimal solutions. Even though `Authentic` instances seem a bit tougher than the previous three sets of instances, our solution method was still able to solve all of them in a short amount of time. Yet it outperformed AP93 and DC15, which did not manage to solve 2 and 34 instances to optimality, respectively, within the imposed time limits. As seen previously, the complexity of the problem increases with the number of stacks.

Note that all instances of the first four sets did not require any shift. The heuristic of Ding and Chou (2015) already showed that the number of shifts is always very small compared to the number of transported containers and it usually decreases when the number of stacks increases. Even though we cannot prove that a solution with no shifts always exists for these types of transportation matrices and we could not address instances with more than 10 ports, we can expect that even if we increase n and $\bar{c}$, an optimal solution with no shift will likely still exist.

A summary of the computational results achieved on the new set of `Required` instances is given in Table 6 (detailed results are provided in Tables EC.13–EC.15 in the online appendix). These instances are much harder to solve than the other four sets: our method could solve 64 of the 81 instances, whereas AP93 and DC15 could solve 28 and 8 instances only. Therefore, our method still outperforms the two compact formulations. The lower bounding procedure has small computing times of a few seconds and provides strictly positive lower bounds; in many cases, the combinatorial lower bound coincides with the lower bound provided by the linear relaxation of the new formulation. When comparing the lower bounds of the three methods, we noted that

Table 5. Summary of the Computational Results on `Authentic` Instances

			AP93					DC15					This paper							
n	$\bar{r}$	$\bar{c}$	LB	UB	Opt	T_{ub}	T_{tot}	LB	UB	Opt	T_{ub}	T_{tot}	LB_0	Vars	Iter	LB	T_{lb}	UB	Opt	T_{tot}
6	6	100	0	0	3	1	1	0	0	3	8	8	0	312	5	0	0	0	3	0
6	6	300	0	0	3	2	2	0	0	3	18	18	0	516	3	0	0	0	3	0
6	6	500	0	0	3	21	21	0	0	3	262	262	0	707	5	0	0	0	3	1
6	8	100	0	0	3	3	3	0	0	3	28	28	0	516	5	0	0	0	3	0
6	8	300	0	0	3	11	11	0	0	3	267	267	0	718	3	0	0	0	3	1
6	8	500	0	0	3	21	21	0	0	3	1,502	1,502	0	911	4	0	0	0	3	0
6	10	100	0	0	3	4	4	0	0	3	42	42	0	766	6	0	0	0	3	0
6	10	300	0	0	3	39	39	0	0	3	1,266	1,266	0	969	5	0	0	0	3	0
6	10	500	0	0	3	68	68	0	9,326	1	1,194	3,240	0	1,182	5	0	0	0	3	1
8	6	100	0	0	3	13	13	0	0	3	265	265	0	507	12	0	0	0	3	0
8	6	300	0	0	3	50	50	0	0	3	2,303	2,303	0	707	5	0	0	0	3	2
8	6	500	0	0	3	70	70	0	94	1	1,248	2,687	0	897	7	0	0	0	3	6
8	8	100	0	0	3	117	117	0	0	3	574	574	0	906	15	0	0	0	3	0
8	8	300	0	0	3	148	148	0	27	1	3,125	3,571	0	1,095	9	0	0	0	3	3
8	8	500	0	0	3	296	296	0	17,278	0	1,210	3,600	0	1,309	7	0	0	0	3	6
8	10	100	0	0	3	74	74	0	0	3	1,386	1,386	0	1,406	17	0	0	0	3	0
8	10	300	0	0	3	178	178	0	299	0	3,092	3,600	0	1,615	9	0	0	0	3	3
8	10	500	0	0	3	490	490	0	22,471	0	2,337	3,600	0	1,776	5	0	0	0	3	5
10	6	100	0	0	3	76	76	0	0	3	1,710	1,710	0	792	25	0	0	0	3	2
10	6	300	0	1	2	449	1,526	0	3,573	0	2,050	3,600	0	982	18	0	0	0	3	67
10	6	500	0	0	3	1,490	1,490	0	18,071	0	2,729	3,600	0	1,135	7	0	0	0	3	34
10	8	100	0	0	3	117	117	0	0	3	3,319	3,319	0	1,426	28	0	0	0	3	3
10	8	300	0	0	3	1,781	1,781	0	15,088	0	1,511	3,600	0	1,603	16	0	0	0	3	24
10	8	500	0	0	3	2,170	2,170	0	24,404	0	3,600	3,600	0	1,793	12	0	0	0	3	28
10	10	100	0	0	3	177	177	0	8	2	3,178	3,349	0	2,297	32	0	1	0	3	7
10	10	300	0	0	3	1,039	1,039	0	18,071	0	1,999	3,600	0	2,391	10	0	1	0	3	74
10	10	500	0	10,176	2	1,562	2,674	0	30,087	0	3,600	3,600	0	2,661	13	0	3	0	3	29
Avg						388	469				1,623	2,155			1,181	11		0		11
Solved					79					47									81	

the linear relaxations of AP93 and DC15 always provide zero shifts, while the proposed method returns a strictly positive number of shifts. Whenever an optimal solution can be found, the number of shifts required is equal to the lower bound returned by the bounding procedure—this cannot be seen from Table 6 but from Tables EC.13–EC.15 in the online appendix. It is interesting to note that, overall, the number of shifts required is always of a few units; moreover, it looks like an increase in the number of ports, stacks, and/or tiers does not necessarily translate into more shifts in the optimal solutions.

8.3. Impact of the Stabilization Technique

In Table 7, we show the impact of the stabilization technique described in Section 6.3 on the lower bounding procedure. We selected 10 `Required` instances with 10 ports (of which we report $\bar{r}$, $\bar{c}$, and the instance number, no) where the final lower bound is strictly greater than the combinatorial lower bound, and we ran the bounding procedure with (stabilized version) and without (nonstabilized version) inequalities (20)–(22). For the two versions, we report the number of iterations (Iter), the final lower bound (LB), and the computing time (T_{lb}). We also indicate the total number of DOIs (inequalities (20) + (21) + (22)) added in the stabilized version of the column generation.

From Table 7, we see that the final lower bound always coincides, thus suggesting that inequalities (20)–(22) may be DOIs even though we do not have a formal proof (as stated in Section 6.3). Moreover, the positive effects of the inequalities in the stabilized version are evident: there is a decrease of 95.2% of the number of column generation iterations required to converge and a decrease of 99.6% of the computing time.

8.4. Impact of the Lower Bounding Procedure

The goal of the lower bounding procedure is to compute the lower bound LB, which is used to choose between looking for a feasible solution with no shifts by solving model (29)–(35) and looking for an optimal solution that allows shifts in no more than LB – n Cont stacks by solving model (36)–(49). Even though the computing time to achieve LB is almost negligible (on average about one second), the time to code the different components of the lower bounding procedure is significant. Therefore, the reader may wonder if it is worth having a procedure to compute LB, or whether the two MIP models (29)–(35) and (36)–(49) could be directly used to find an optimal CSPP solution.

Table 6. Summary of the Computational Results on `Required` Instances

			AP93					DC15					This paper							
n	$\bar{r}$	$\bar{c}$	LB	UB	Opt	T_{ub}	T_{tot}	LB	UB	Opt	T_{ub}	T_{tot}	LB_0	Vars	Iter	LB	T_{lb}	UB	Opt	T_{tot}
6	6	100	2	2	3	38	38	1	2	2	58	1,257	2	284	5	2	0	2	3	0
6	6	300	3	4	2	385	1,583	2	4	1	33	2,418	2	461	5	4	0	4	3	65
6	6	500	2	2	3	1,768	1,768	1	2	0	391	3,600	2	676	3	2	0	2	3	11
6	8	100	2	2	3	174	174	0	2	0	23	3,600	2	476	7	2	0	2	3	3
6	8	300	2	3	2	331	1,527	2	3	1	184	2,551	2	614	4	3	0	3	3	6
6	8	500	1	2	1	903	3,288	0	2	0	542	3,600	2	871	5	2	0	2	3	220
6	10	100	3	3	3	196	195	2	3	2	34	1,229	3	692	8	3	0	3	3	5
6	10	300	2	2	2	687	1,883	0	2	0	856	3,600	2	908	4	2	0	2	3	2
6	10	500	0	2	1	26	2,408	0	3	1	726	2,446	2	1,072	5	2	0	2	3	128
8	6	100	0	2	1	66	2,430	0	2	0	644	3,600	2	495	14	2	0	2	3	132
8	6	300	0	2	1	488	2,874	0	3	0	1,120	3,600	2	700	6	2	0	2	3	47
8	6	500	1	3	1	1,193	3,576	0	43	0	734	3,600	2	852	18	3	0	3	3	840
8	8	100	1	2	1	116	2,408	0	2	0	487	3,600	2	829	18	2	1	2	3	68
8	8	300	0	5	0	1,665	3,600	0	16	0	1,701	3,600	4	1,027	16	4	0	5	2	2,393
8	8	500	0	3	0	323	3,600	0	5,922	0	2,452	3,600	3	1,262	12	3	0	3	1	2,559
8	10	100	1	6	1	739	2,976	0	6	0	2,806	3,600	4	1,182	17	6	1	6	2	1,270
8	10	300	0	6	0	458	3,600	0	163	0	574	3,600	3	1,350	19	5	1	7	2	1,270
8	10	500	0	7,332	0	597	3,600	0	7,318	0	1,977	3,600	2	1,647	20	4	1	5	2	2,299
10	6	100	1	1	2	417	1,348	0	3	1	1,347	2,533	1	754	25	1	0	1	3	28
10	6	300	1	5	0	312	3,600	0	94	0	1,417	3,600	2	917	37	4	1	4	2	1,383
10	6	500	0	10,735	0	252	3,600	0	10,978	0	1,863	3,600	2	1,143	38	3	0	5	1	2,459
10	8	100	0	7	0	1,762	3,600	0	34	0	2,727	3,600	4	1,357	39	4	1	5	1	2,424
10	8	300	1	5,237	0	472	3,600	0	9,645	0	1,620	3,600	2	1,583	21	2	3	3	2	1,814
10	8	500	0	8,559	0	382	3,600	0	23,720	0	2,735	3,600	2	1,666	13	3	1	*3	2	2,125
10	10	100	0	5	0	1,656	3,600	0	25	0	2,415	3,600	3	1,824	32	5	3	6	2	1,506
10	10	300	0	6,100	1	253	2,599	0	16,533	0	1,246	3,600	3	2,205	38	4	5	5	2	1,747
10	10	500	0	3	0	1,427	3,600	0	28,619	0	3,600	3,600	2	2,408	27	3	5	*5	1	2,416
Avg						633	2,618				1,271	3,260		1,084	17		1			1,008
Solved					28					8									64	

Note. An asterisk indicates that in one of the instances no feasible solution was found.

Table 7. Effects, on a Sample Set of `Required` Instances with 10 Ports, of Stabilizing the Bounding Procedure

			Nonstabilized			Stabilized			
$\bar{r}$	$\bar{c}$	No	Iter	LB	T_{lb}	nDOI	Iter	LB	T_{lb}
6	300	2	143	4	3	480	20	4	1
6	300	3	501	5	18	475	78	5	2
6	500	2	955	5	25	459	93	5	1
8	300	3	1,431	3	101	1,202	37	3	5
8	500	1	338	4	12	968	9	4	1
8	500	2	561	2	29	1,105	20	2	3
10	100	1	282	6	7	1,255	27	6	2
10	100	2	798	5	69	1,782	52	5	4
10	300	2	2,641	8	6,655	1,564	86	8	5
10	500	1	2,069	5	341	1,625	45	5	4
Avg			972		726		47		3
Saving (%)							95.2		99.6

A simple approach (hereafter called NoLB) to find an optimal CSPP could be to first solve model (29)–(35) and then, if a solution with no shifts does not exist, solve model (36)–(49) by removing constraint (50) and by allowing shifts on a limited but large enough set of stacks—for example, by setting $|C_Y| = 10$. Based on the computational results reported in Tables 2–6, this simple approach should be able to find an optimal solution on all instances without having to compute LB.

The computational results achieved on the `Long`, `Mixed`, `Short`, and `Authentic` by the NoLB approach would clearly be the same as that already reported in Tables 2–5 because the computing time taken by the lower bounding procedure is on average less than a second and a feasible solution with no shifts exists on all these instances.

In Table 8, we compare the computational results achieved on the `Required` instances by the exact method proposed in this paper and by the NoLB approach. The first five columns report the same information as that already reported in Table 6 whereas the four columns under the heading NoLB indicate the computing time to solve model (29)–(35) (T_0), the number of instances solved to optimality (Opt), the computing time to solve model (36)–(49) (T_{10}), and the total computing time (T_{tot}), where $T_{tot} = T_0 + T_{10}$. A time limit of one hour was imposed on each model.

Table 8 clearly shows the benefits of computing the lower bound LB. The proposed exact method that computes LB could solve 64 of the 81 `Required` instances whereas the NoLB approach could close only nine instances. We also note that the average computing time

Table 8. Computational Results on `Required` Instances With and Without Computing Lower Bound LB

			With LB		NoLB			
n	$\bar{r}$	$\bar{c}$	Opt	T_{tot}	T_0	Opt	T_{10}	T_{tot}
6	6	100	3	0	0	2	1,458	1,458
6	6	300	3	65	556	2	1,256	1,813
6	6	500	3	11	2,478	0	3,600	6,078
6	8	100	3	3	9	0	3,600	3,609
6	8	300	3	6	239	1	2,406	2,645
6	8	500	3	220	1,222	0	3,600	4,822
6	10	100	3	5	37	2	1,614	1,651
6	10	300	3	2	1,987	0	3,600	5,587
6	10	500	3	128	79	0	3,600	3,679
8	6	100	3	132	620	0	3,600	4,220
8	6	300	3	47	3,600	0	3,600	7,200
8	6	500	3	840	2,402	1	2,950	5,351
8	8	100	3	68	1,268	0	3,600	4,868
8	8	300	2	2,393	3,600	0	3,600	7,200
8	8	500	1	2,559	3,600	0	3,600	7,200
8	10	100	2	1,270	2,163	0	3,600	5,763
8	10	300	2	1,270	2,401	0	3,600	6,001
8	10	500	2	2,299	3,600	0	3,600	7,200
10	6	100	3	28	2,400	1	2,414	4,814
10	6	300	2	1,383	3,600	0	3,600	7,200
10	6	500	1	2,459	2,430	0	3,600	6,030
10	8	100	1	2,424	1,434	0	3,600	5,034
10	8	300	2	1,814	3,600	0	3,600	7,200
10	8	500	2	2,125	3,600	0	3,600	7,200
10	10	100	2	1,506	1,882	0	3,600	5,482
10	10	300	2	1,747	2,407	0	3,600	6,007
10	10	500	1	2,416	3,600	0	3,600	7,200
Avg				1,008	2,030		3,248	5,278
Solved			64			9		

to prove that no feasible solution with no shifts exists is already twice as much as the time to find the optimal solution. Moreover, we can see that the computing time to solve model (36)–(49) by setting $|C_Y| = 10$ is significantly higher than the time to find an optimal solution with the proposed exact method.

9. Conclusions

We have considered the *container stowage planning problem* (CSPP) where the number of loading and unloading movements of containers stowed in a ship visiting a predefined ordered sequence of ports has to be minimized. The CSPP has been previously addressed with heuristics and compact *mixed-integer programming* models. Nonetheless, finding an optimal solution or even a good lower bound is an open problem even for small-sized instances.

We have introduced a combinatorial lower bound and a novel MIP model whose linear relaxation provides high-quality lower bounds. An efficient bounding procedure to compute such lower bounds has been proposed. The bounding procedure has been embedded into a simple yet effective solution method to find an optimal stowage plan.

Computational results have shown that the proposed solution method outperforms the methods available from the literature and that the optimal solutions of test instances with up to 10 ports and 5,000 TEU can be found in a few minutes of computing time.

Many operational constraints arising in real-life CSPPs have not been considered in the basic CSPP addressed in this paper, but we believe that the proposed formulation and the solution method could be successfully extended to handle a wide variety of operational constraints. The feasibility of stack plans can be defined to handle, for example, containers of different size, weight, and type, stacks of different size, the fact that ships are never empty, and other operational constraints related to a single stack; moreover, the feasibility of port layouts can be defined to handle, for example, ship stability constraints and other constraints involving multiple bays or stacks. Changing the definition of stack plans and port layout would naturally require to properly adapt the different algorithms to solve the pricing problems to use the proposed formulation (or some extensions of it) to find optimal stowage plans. Investigating all these possible extensions of the proposed formulation and solution method is beyond the scope of this paper, but we consider it as a promising avenue for future research on the topic. We also believe that the insight on the basic CSPP provided by this paper can be of inspiration to solve stowage planning problems similar to the problems faced by stowage planners in practice.

Acknowledgments

The authors are grateful to the anonymous referees for their comments that improved the presentation of the paper.

References

Alves C, Valerio de Carvalho JM (2008) A stabilized branch-and-price-and-cut algorithm for the multiple length cutting stock problem. *Comput. Oper. Res.* 35(4):1315–1328.

Ambrosino D, Paolucci M, Sciomachen A (2015) Experimental evaluation of mixed integer programming models for the multi-port master bay plan problem. *Flexible Services Manufacturing J.* 27(2–3):–284.

Ambrosino D, Paolucci M, Sciomachen A (2017) Computational evaluation of a MIP model for multi-port stowage planning problems. *Soft Comput.* 21(7):1753–1763.

Ambrosino D, Sciomachen A, Tanfani E (2004) Stowing a containership: The master bay plan problem. *Transportation Res. Part A: Policy Practice* 38(2):81–99.

Ambrosino D, Sciomachen A, Tanfani E (2006) A decomposition heuristics for the container ship stowage problem. *J. Heuristics* 12(3):211–233.

Ambrosino D, Anghinolfi D, Paolucci M, Sciomachen A (2009) A new three-step heuristic for the master bay plan problem. *Maritime Econom. Logist.* 11(1):98–120.

Aslidis A (1990) Minimizing of overstowage in containership operations. *Oper. Res.* 90:457–471.

Avriel M, Penn M (1993) Exact and approximate solutions of the container ship stowage problem. *Comput. Indust. Engrg.* 25(1–4): 271–274.

Avriel M, Penn M, Shpirer N (2000) Container ship stowage problem: Complexity and connection to the coloring of circle graphs. *Discrete Appl. Math.* 103(1–3):271–279.

Avriel M, Penn M, Shpirer N, Witteboon S (1998) Stowage planning for container ships to reduce the number of shifts. *Ann. Oper. Res.* 76(1–4):55–71.

Ben Amor H, Desrosiers J, Valerio de Carvalho JM (2006) Dual-optimal inequalities for stabilized column generation. *Oper. Res.* 54(3):454–463.

Botter RC, Brinati MA (1992) Stowage container planning: A model for getting an optimal solution. Vieria CB, Martins P, Kuo C, eds. *Proc. IFIP TC5/WG5.6 Seventh Internat. Conf. Comput. Appl. Automation Shipyard Operation Ship Design, VII* (North-Holland, Amsterdam), 217–229.

Clautiaux F, Alves C, Valerio de Carvalho JM, Rietz J (2011) New stabilization procedures for the cutting stock problem. *INFORMS J. Comput.* 23(4):530–545.

Delgado A, Jensen RM, Janstrup K, Trine Høyer R, Andersen KH (2012) A constraint programming model for fast optimal stowage of container vessel bays. *Eur. J. Oper. Res.* 220(1): 251–261.

Ding D, Chou MC (2015) Stowage planning for container ships: A heuristic algorithm to reduce the number of shifts. *Eur. J. Oper. Res.* 246(1):242–249.

Dubrovsky O, Levitin G, Penn M (2002) A genetic algorithm with a compact solution encoding for the container ship stowage problem. *J. Heuristics* 8(6):585–599.

Kang JG, Kim YD (2002) Stowage planning in maritime container transportation. *J. Oper. Res. Soc.* 53(4):415–426.

Levison M (2010) *The Box: How the Shipping Container Made the World Smaller and the World Economy Bigger* (Princeton University Press, Princeton, NJ).

Pacino D, Jensen RM (2013) Fast slot planning using constraint-based local search. Yang G-C, Ao S-I, Huang X, Castillo O, eds. *IAENG Transactions on Engineering Technologies: Special Issue of the International Multi Conference of Engineers and Computer Scientists 2012* (Springer, Dordrecht, Netherlands), 49–63.

Pacino D, Delgado A, Jensen RM, Bebbington T (2011) Fast generation of near-optimal plans for eco-efficient stowage of large container vessels. Böse JW, Hu H, Jahn C, Shi X, Stahlbock R, Voß S, eds. *Comput. Logist.* (*ICCL11*), Lecture Notes Comput. Sci., Vol. 6971 (Springer, Berlin Heidelberg), 286–301.

Pacino D, Delgado A, Jensen RM, Bebbington T (2012) An accurate model for seaworthy container vessel stowage planning with ballast tanks. Hu H, Shi X, Stahlbock R, Voß S, eds. *Comput. Logist.* (*ICCL12*), Lecture Notes Comput. Sci., Vol. 7555 (Springer, Berlin Heidelberg), 17–32.

Sciomachen A, Tanfani E (2003) The master bay plan problem: A solution method based on its connection to the three dimensional bin packing problem. *IMA J. Management Math.* 14(3):251–269.

Sciomachen A, Tanfani E (2007) A 3D-BPP approach for optimising stowage plans and terminal productivity. *Eur. J. Oper. Res.* 183(3):1433–1446.

Tierney K, Pacino D, Jensen RM (2014) On the complexity of container stowage planning problems. *Discrete Appl. Math.* 169(May):225–230.

UNCTAD (2015) Review of maritime transport 2014. United Nations Conference on Trade and Development, Geneva.

Wilson ID, Roach PA (1999) Principles of combinatorial optimization applied to container-ship stowage planning. *J. Heuristics* 5(4):403–418.